UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

PLANTARUM: UMA APLICAÇÃO ANDROID PARA CONSULTAS DE PLANTAS

MATHEUS BORTOLON

BLUMENAU
2014

2014/1-17

MATHEUS BORTOLON

PLANTARUM: UMA APLICAÇÃO ANDROID PARA CONSULTAS DE PLANTAS

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciência da Computação — Bacharelado.

Prof. Aurélio Faustino Hoppe, Mestre - Orientador

BLUMENAU
2014

2014/1-17

PLANTARUM: UMA APLICAÇÃO ANDROID PARA CONSULTAS DE PLANTAS

Por

MATHEUS BORTOLON

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

	_____
Presidente:	Prof. Aurélio Faustino Hoppe, Mestre – Orientador, FURB
	_____
Membro:	Prof. Dalton Solano dos Reis, Mestre – FURB
	_____
Membro:	Prof. Francisco Adell Péricas, Mestre – FURB

Blumenau, 09 de julho de 2014

Dedico este trabalho à minha família, que me ofereceram sempre seu apoio e motivação para a conclusão deste trabalho.

AGRADECIMENTOS

Dedico este trabalho a Deus que fez de mim o que sou hoje.

À minha família por toda a ajuda e paciência. Agradeço especialmente a minha mãe Sandra Maria Bortolon pela ajuda em tornar o que era um sonho em uma realidade.

Aos meus amigos, por me apontarem a direção do conhecimento necessário para a execução deste trabalho.

Em especial à Arno Wilson Cassaniga por ter iniciado este trabalho, e principalmente pela ajuda durante o desenvolvimento.

À minha namorada pela compreensão e apoio nos momentos necessários.

Ao meu orientador, Aurélio Faustino Hoppe, por ter idealizado e acreditado na viabilidade deste trabalho.

Talento é 1% inspiração e 99% transpiração.

Thomas Edison

RESUMO

Este trabalho apresenta o desenvolvimento de uma aplicação móvel capaz de interagir com um serviço web para o cadastro e classificação de espécies de plantas através da captura de imagens pelo dispositivo móvel onde a aplicação está sendo executada. As imagens capturadas são enviadas juntamente com detalhes de captura e três características macromorfológicas para um servidor web que, por meio de uma API, é capaz de cadastrar ou classificar a amostra em uma base de dados. Os resultados obtidos tornam possível o reconhecimento e classificação de espécies baseando-se no registro de folhas.

Palavras-chave: Classificação de plantas. Dispositivos móveis. Android.

ABSTRACT

This work presents the development of a mobile application capable of interacting with a web service for registration and classification of plant species by capturing images with a mobile device where the application is running. Captured images are sent along with details of capture and three macromorphological features to a web server that, through an API, is capable of registering or classify the sample into a database. The results make possible the recognition and classification of species based on the leaves database.

Key-words: Plants classify. Mobile devices. Android.

LISTA DE ILUSTRAÇÕES

Figura 1 – Exemplos de tipos de folhas 15
Figura 2 – Exemplo de folha discolor 16
Figura 3 – Funcionalidades da aplicação Leafsnap 19
Figura 4 – a) Visualização da detecção de bordas b) interface de rotação da imagem 21
Figura 5 – Processo de captura da imagem. 23
Figura 6 – Resultado do cálculo de distância euclidiana 23
Quadro 1 – Características dos trabalhos relacionados 24
Figura 7 – Diagrama de casos de uso 26
Figura 8 – Pacote `Activities` 27
Figura 9– Pacote `Utils` 29
Figura 10 – Diagrama de classes que representa o serviço WCF do servidor 30
Figura 11 – Diagrama de atividades para cadastrar amostra 31
Figura 12 – Diagrama de atividades para classificação de espécies 32
Quadro 2 – Iniciando a `Intent` para acessar a câmera 34
Quadro 3 – Classe `GPS` 35
Quadro 4 – Implementação do método `setGPSInfo` 36
Quadro 5 – Convertendo coordenadas da tela para coordenadas da imagem 37
Quadro 6 – Salvando informações na imagem 39
Quadro 7 – Salvando a imagem no dispositivo 40
Quadro 8 – Salvando a imagem no dispositivo 41
Quadro 9 – Enviando a imagem para classificação 42
Quadro 10 – Exibindo o resultado da classificação 43
Quadro 11 – Contrato da aplicação do servidor 44
Quadro 12 – Retornando os nomes das espécies da base de dados 45
Quadro 13 – Recebendo uma imagem para classificação 46
Quadro 14 – Conectando com a API para a classificação 47
Quadro 15 – Recebendo uma imagem cadastro de amostra ou espécie 49
Quadro 16 – Cadastrando uma nova amostra ou nova espécie 50
Figura 13 – Tela inicial da aplicação 52
Figura 14 – a) Fotografar amostra; b) confirmar o resultado 53
Figura 15 – Informações da amostra 53

Figura 16 – Informação do pecíolo e ponta da amostra 54
Figura 17 – Resultado com sucesso da classificação 55
Figura 18 – Tela de cadastro de espécies 56
Figura 19 – Tela para seleção de arquivo 57
Figura 20 – Configuração da base de dados para teste 59
Figura 21 – Espécies cadastradas para testes reais 61
Quadro 16 – Comparativo dos trabalhos correlatos com este projeto 62

LISTA DE TABELAS

Tabela 1 – Espécies de plantas utilizadas na validação da aplicação 60
Tabela 2 – Comparação dos resultados de classificação entre as versões da API 60

LISTA DE SIGLAS

ADT – *Android Developer Tools*

API – *Application Programming Interface*

DLL – *Dinamic Link Libary*

EXIF – *Exchangeable Image File Format*

GPS – *Global Positionning System*

HTTP – *Hypertext Transfer Protocol*

IDE – *Integrated Development Enviroment*

JSE – *Java Standard Edition*

NDC – *Normalized Device Coordinatest*

SDK – *Software Development Kit*

UML – *Unified Modeling Language*

WCF – *Windows Communication Foundation*

XML – *Extensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO 13
1.1 OBJETIVOS DO TRABALHO 13
1.2 ESTRUTURA DO TRABALHO 14
2 FUNDAMENTAÇÃO TEÓRICA 15
2.1 MORFOLOGIA EXTERNA DAS PLANTAS 15
2.2 DESENVOLVIMENTO DE WEBSERVICES 16
2.3 API PLANTARUM 17
2.4 TRABALHOS CORRELATOS 18
2.4.1 Leafsnap: A computer vison for automatic plant species identification 18
2.4.2 Automatic plant leaf classification for a mobile field guide: an Android aplication 20
2.4.3 Shape based plant leaf classification system using Android 22
2.4.4 Seção comparativa dos trabalhos relacionados 24
3 DESENVOLVIMENTO 25
3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO 25
3.2 ESPECIFICAÇÃO 25
3.2.1 Diagrama de casos de uso 25
3.2.2 Diagrama de classes 26
3.2.2.1 Diagramas do aplicativo cliente 27
3.2.2.2 Diagrama de classes do aplicativo servidor 29
3.2.3 Diagramas de sequencia 30
3.3 IMPLEMENTAÇÃO 32
3.3.1 Técnicas e ferramentas utilizadas 32
3.3.1.1 Implementação da aplicação cliente 33
3.3.1.2 Implementação da aplicação servidor 43
3.3.1.3 Implementação de novas características na API 51
3.3.2 Operacionalidade da implementação 51
3.4 RESULTADOS E DISCUSSÃO 58
3.4.1 Testes com banco de imagens 58
3.4.2 Testes com espécies locais 61
3.4.3 Comparação com os trabalhos correlatos 62
4 CONCLUSÕES 64

4.1 LIMITAÇÕES 65
4.2 EXTENSÕES 65
REFERÊNCIAS 66
APÊNDICE A – Imagens utilizadas para a validação da aplicação. 68

1 INTRODUÇÃO

Atualmente o processo de registro e classificação de plantas em uma escala global é feito quase que na sua totalidade de forma manual, onde são recolhidas amostras e o profissional de botânica informa as características para a catalogação desta espécie. Segundo Plotze (2004, p. 6) o processo de catálogo e classificação de plantas é demasiado trabalhoso, pois neste processo o profissional de botânica colhe as amostras, desseca-as, prensa-as e descreve as características da espécie em uma ficha padronizada para anexar com a amostra no catálogo.

Segundo Plotze (2004, p. 6), o botânico deve possuir grandes conhecimentos e habilidades na área de taxonomia para ser capaz de fazer a identificação de espécies. Por ser todo executado manualmente e possuir alto grau de complexidade, a probabilidade de acontecer algum erro humano é muito alta. Ao omitir alguma característica da planta, o botânico pode estar comprometendo todo o processo de classificação desta espécie.

O desenvolvimento de tecnologias voltadas à área de catalogação e classificação de espécies vegetais é uma necessidade que pode ser suprida com a construção de ferramentas para auxiliar os profissionais de botânica nestas tarefas (PLOTZE, 2004, p. 2). Visando reduzir os problemas causados pela deterioração e perda de características relevantes para a classificação de espécies, tais ferramentas poderiam armazenar as amostras de maneira digital em um banco de imagens.

Neste cenário, acredita-se que a implementação de uma ferramenta para auxiliar os botânicos no cadastro e classificação de espécies vegetais, seria uma forma de reduzir estes problemas gerados pelo processo manual. Assim surge a motivação de criar uma aplicação para tal. A partir de um catálogo digital, o reconhecimento de espécies vegetais pode ser automatizado, pois as informações para a comparação estarão definidas e armazenadas digitalmente, livres de deterioração.

O trabalho proposto desenvolve uma aplicação *mobile* capaz de armazenar/coletar informações de plantas no seu ambiente natural, tais como: imagens, características e sua localização. Estas informações são enviadas para um servidor a fim de catalogar ou obter a classificação da espécie.

1.1 OBJETIVOS DO TRABALHO

O objetivo geral deste trabalho é disponibilizar uma aplicação para Android capaz de interagir com um *webservice* para efetuar o cadastro e classificação de espécies de plantas.

Os objetivos específicos são:

a) disponibilizar uma aplicação para dispositivos móveis que permita capturar imagens a partir da câmera do dispositivo;
b) prover um serviço *Windows Communication Foundation* (WCF) para realizar a troca de informações entre o dispositivo móvel e a API Plantarum de Cassaniga (2012);
c) fornecer um mecanismo de comunicação e armazenamento assíncrono para que na indisponibilidade de conexão com o webservice, as amostras, a data e hora e as coordenadas de *Global Positioning System* (GPS) das plantas sejam armazenadas no próprio dispositivo móvel para envio posterior;
d) oferecer uma interface gráfica no dispositivo móvel para exibir as informações retornadas pela API Plantarum de Cassaniga (2012).

1.2 ESTRUTURA DO TRABALHO

O primeiro capítulo apresenta a motivação e os objetivos a serem atingidos com este trabalho.

O segundo capítulo contém a base teórica dos principais conceitos e técnicas exploradas no desenvolvimento do projeto.

No terceiro capítulo estão descritos a arquitetura, a implementação e os resultados obtidos nos testes de validação da aplicação.

O quarto capítulo contém as conclusões formuladas a partir do trabalho e sugestões para extensões ao mesmo.

2 FUNDAMENTAÇÃO TEÓRICA

Nas próximas seções são apresentados os principais conceitos relacionados ao processo de desenvolvimento de aplicações para dispositivos móveis e webservices. Na seção 2.1 é explicado sobre morfologia externa das folhas. A seção 2.2 expõe o desenvolvimento de webservices. Na seção 2.3 é demonstrada a API Plantarum de Cassaniga (2012). Por fim, a seção 2.4 apresenta três trabalhos correlatos e um comparativo entre eles.

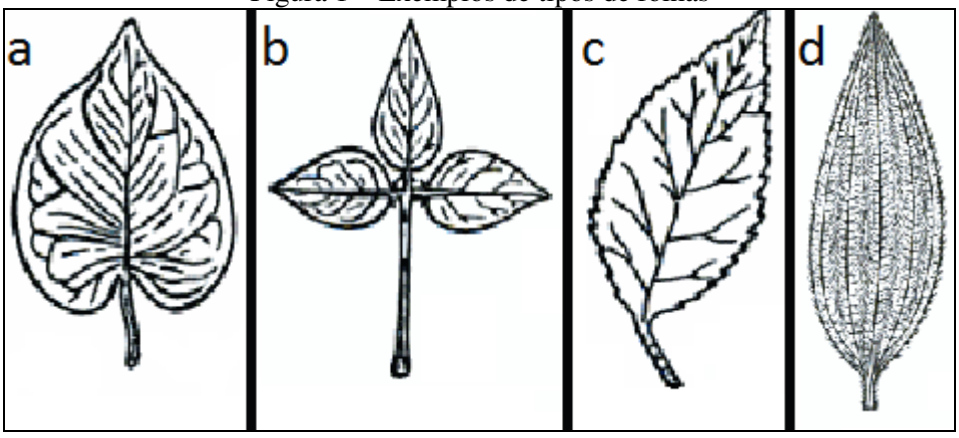
2.1 MORFOLOGIA EXTERNA DAS PLANTAS

Segundo Wiggers e Stange (2008, p. 3) para a classificação de plantas a taxonomia baseia-se em caracteres da morfologia floral e da anatomia dentre outros. A identificação das espécies é essencial para o desenvolvimento das ciências básicas e aplicadas.

Ferri (1981 p. 49) explica que a morfologia de folhas pode variar por inúmeras causas, principalmente por adaptação ao modo e local onde se encontram. Já a morfologia do restante da planta depende principalmente da distribuição de suas folhas.

Adaptações morfológicas em uma folha têm a função de fazer com que ela tenha condições de produzir material orgânico para sua sobrevivência. Segundo Ferri (1981, p. 52) uma folha pode ter formas e superfícies diferentes como piloso ou liso, ou também ser inteiro ou com divisões também chamados de simples e composto respectivamente. A Figura 1 exibe os tipos de superfície e os tipos simples e composto.

Figura 1 – Exemplos de tipos de folhas



Fonte: adaptado de Ferri (1981, p. 43-47).

A Figura 1 mostra em ordem: a) folha simples; b) folha composta; c) folha lisa; d) folha pilosa. Segundo Wiggers e Stange (2008, p. 5) o processo de identificação mais comum é por meio da comparação da amostra recém coletada com outra já catalogada.

Wiggers e Stange (2008, p. 33) ainda cita outra característica que deve ser levada em consideração para o reconhecimento, a cor das faces da folha. Algumas espécies apresentam

variação na cor das faces de suas folhas, estas espécies são denominadas discolores. A Figura 2 mostra o exemplo de uma folha discolor.

Figura 2 – Exemplo de folha discolor

Fonte: Wiggers e Stange (2008, p. 33).

Wiggers e Stange (2008, p. 5) explica que classificação é a ordenação das plantas em níveis hierarquicos, de acordo com as características apresentadas.

2.2 DESENVOLVIMENTO DE WEBSERVICES

Segundo Borges (2005, p. 1), webservices podem ser considerados componentes de software independente de plataforma e implementação, unindo a infraestrutura da internet e comunicação via *Extensible Markup Language* (XML). Os webservices são compostos por um fragmento de informação que pode ser acessado de qualquer tipo de dispositivo por qualquer um em qualquer lugar.

Partindo da necessidade da comunicação entre sistemas de naturezas diferentes com uma difícil abertura para comunicação, Gomes (2011) explica que é possível utilizar webservices para resolver este problema, pois permite que duas aplicações desenvolvidas em linguagens diferentes possam comunicar-se de forma simplificada por meio de serviços através de XML.

Ferreira (2012, p. 28) descreve webservices como sistemas autossuficientes e auto descritos que podem ser utilizados por outras aplicações ou serviços. Ferreira (2012) explica que a grande vantagem em fazer uso deste tipo de sistema é poder utilizar as necessidades de um nicho de usuários como resposta para outro nicho de usuários. Assim tem-se o reaproveitamento de código e uma otimização dos serviços conforme sua utilização.

Segundo Borges (2005, p. 1), as vantagens do uso desta tecnologia acontecem por conta da ampla gama de aplicações que é possível se desenvolver a partir dos serviços disponibilizados, não sendo necessário que os sistemas possuam a mesma plataforma ou comunicação direta entre si. Isso abre uma grande possibilidade de extensão de maneira transparente.

Segundo Ferreira (2012, p. 22-23), os princípios fundamentais para esta arquitetura orientada a serviço são visibilidade, interação e eficácia. Neste cenário o paradigma orientado a serviços em que os webservices estão inclusos proporciona algumas vantagens, como cada um de seus serviços serem independentes e reutilizáveis. Os serviços também possuem a vantagem do encapsulamento escondendo sua complexidade e proporcionando um bom nível de segurança. Em complemento, Ferreira (2012, p. 23) explica que os serviços são escaláveis, proporcionando uma redução de custos na cooperação entre sistemas por conta da facilitação da comunicação e gerência de sistemas complexos.

2.3 API PLANTARUM

A API desenvolvida por Cassaniga (2012) possui as funcionalidades de criação de base de dados para o catalogo de plantas, adição de amostras de folhas na base e a classificação de uma amostra. Ao criar uma nova base de dados é possível selecionar o método de classificação que será utilizado para as amostras desta base, as características para discriminar as amostras e também quais serão os valores para os parâmetros utilizados pelo método classificador.

Ao adicionar uma imagem na base de dados devem ser informados onde fica a base e a ponta da folha. Também é possível escolher a espécie da planta se ela já estiver incluída no sistema. Ainda é possível cadastrar um rótulo que pode ser utilizado para identificação. Caso a espécie ainda não possua cadastro na base de dados, é possível fazer este cadastro para posteriormente associar a espécie a outra amostra ou mesmo fazer o reconhecimento desta espécie.

Para a classificação de plantas é necessário que seja passada uma imagem para a API e que seja informado onde fica a base e a ponta da folha. Neste processo Cassaniga (2012) utiliza das imagens as seguintes características:

a) circularidade: define o quanto uma forma geométrica assemelha-se a um círculo;
b) dispersão: representa para a API de Cassaniga (2012) a razão entre a maior e a menor distância do centro até um ponto no contorno da folha;
c) magreza: é a razão entre o comprimento e a largura da folha;
d) descritores de Fourier: são coeficientes resultantes da transformação discreta de Fourier;
e) máscara de nervura: é uma característica que representa a área total que as nervuras ocupam na folha;
f) lacunaridade: representa uma média que é aplicada a textura das plantas, a qual

indica se um padrão aplicado a uma subparte da textura possui poucas ou muitas lacunas;

g) estatísticas de cor: correspondem a cálculos exclusivamente aplicados aos valores do padrão de cor da imagem. São cálculos estatísticos de média, desvio padrão, assimetria e pico ou achatamento na curva.

A API é disponibilizada em formato *Dynamic Link Library* (DLL), facilitando a integração com plataformas independentes e permitindo a sua utilização em variados tipos de projetos.

2.4 TRABALHOS CORRELATOS

Esta seção destina-se a apresentar três trabalhos voltados ao reconhecimento de plantas utilizando dispositivos móveis para a captura da imagem e exibição dos resultados.

2.4.1 Leafsnap: A computer vison for automatic plant species identification

O trabalho desenvolvido por Kumar et al. (2012) descreve uma aplicação que auxilia os usuários a identificar espécies pela fotografia de suas folhas. Para seu trabalho, é necessário que uma espécie de folha seja fotografada com um fundo de cor sólida e clara. O processo de reconhecimento é dividido em quatro etapas:

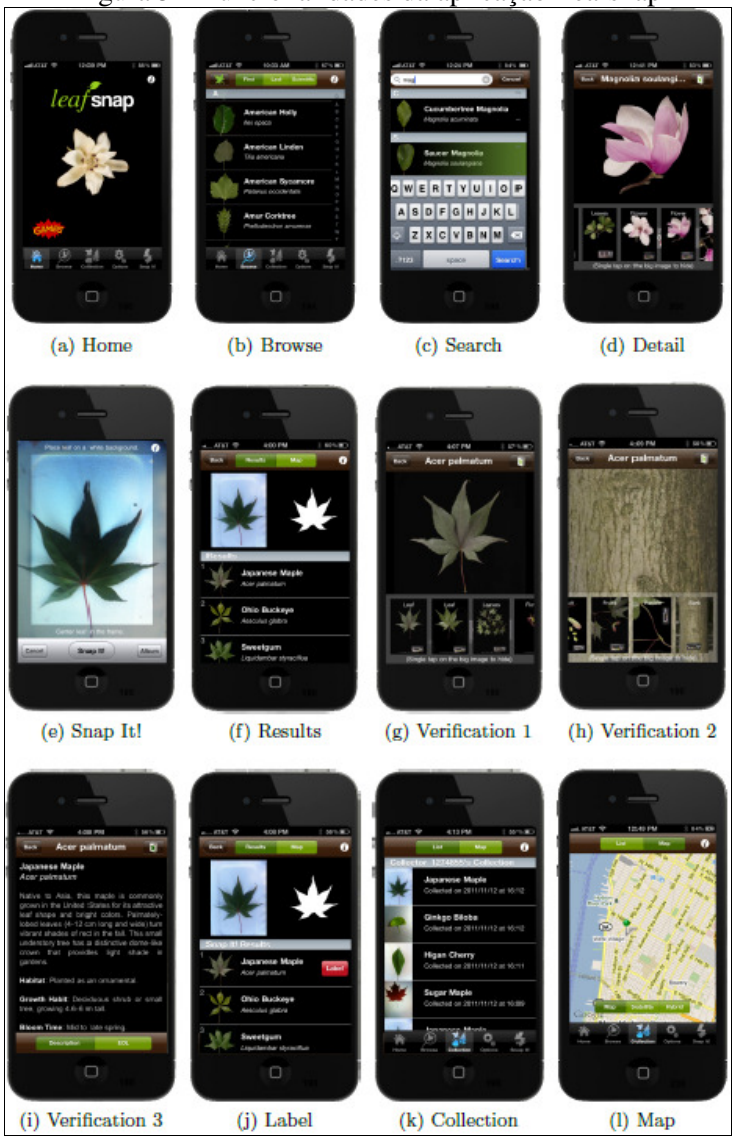
a) classificando: processo onde a imagem é testada para saber se uma folha é válida. Este passo decide se o restante do processamento será feito ou não;
b) segmentando: neste passo acontece a binarização da imagem separando a folha do plano de fundo;
c) extraindo: são extraídas características de curvatura da imagem binarizada para compactamente e descritivamente representar a forma da folha. Neste passo são processados histogramas de curvatura sobre múltiplos tamanhos, utilizando medidas integrais de curvatura;
d) comparando: acontece a comparação das características extraídas da imagem com as características já classificadas da base de dados, a fim de encontrar as correspondências. Devido ao tamanho do conjunto de dados da base, e a capacidade de discriminação das características extraídas, é utilizada a abordagem de vizinhança mais próxima com interseção histograma como a métrica de distância para a comparação.

O tempo total de processamento de todo o ciclo leva em torno de cinco segundos e ao final é exibido ao usuário as melhores combinações encontradas na base de dados. Neste processo quem determina a verdadeira classificação é o usuário da aplicação.

A aplicação desenvolvida por Kumar et al. (2012, p. 11) é composta por um sistema *end-end* onde a imagem de entrada é adquirida em dispositivos móveis e enviada ao servidor, onde acontece a extração de características e a busca de correspondências. Ao final o servidor retorna o resultado das correspondências para exibição no dispositivo móvel.

A Figura 3 demonstra algumas funcionalidades da aplicação móvel de Kumar et al. (2012).

Figura 3 – Funcionalidades da aplicação Leafsnap



Fonte: Kumar et al. (2012, p. 12).

Segundo Kumar et al.(2012, p. 11) a Figura 1 é composta de: (a) tela inicial, com uma escolha aleatória de imagem para ser exibida, (b) tela de navegação com uma lista ordenável de espécies, com a funcionalidade de busca para as espécies contidas no sistema, (c) funcionalidade de busca para encontrar uma espécie em particular por seu nome comum ou científico, (d) visão detalhada de uma espécie em particular, mostrando diferentes imagens cadastradas para uma mesma espécie, (e) tela *Snap it!*, que possibilita a identificação automática, (f) resultado da identificação automática, (g-i) verificação manual, onde o usuário navega nas características das espécies retornadas da identificação, (j) rotulando a correspondência correta (k) adição da folha para a coleção local, (l) um mapa do local onde a folha foi coletada.

No trabalho de Kumar et al. (2012, p. 13), é citado que o servidor de *bakend* é um *Intel Xeon* com dois processadores quad core rodando a 2.33 Ghz cada, e 16 *GB* de *RAM*. Com exceção de algumas versões de imagens em alta resolução, que são servidas via *Amazon's Simple Storage Service* (S3), todas as outras operações são gerenciadas pelo servidor deles.

2.4.2 Automatic plant leaf classification for a mobile field guide: an Android aplication

Knight, Painter e Potter (2010) desenvolveram uma aplicação que permite ao usuário identificar espécies vegetais através de fotografias das folhas das plantas, tiradas por um *smartphone* equipado com o sistema operacional Android. Os autores ressaltam o uso da plataforma Android pelo fato da maioria destes dispositivos serem equipados com uma câmera de alta qualidade, possuindo *flash* e foco automático.

Em mais alto nível o algoritmo desenvolvido por Knight, Painter e Potter (2010) segue as seguintes etapas:

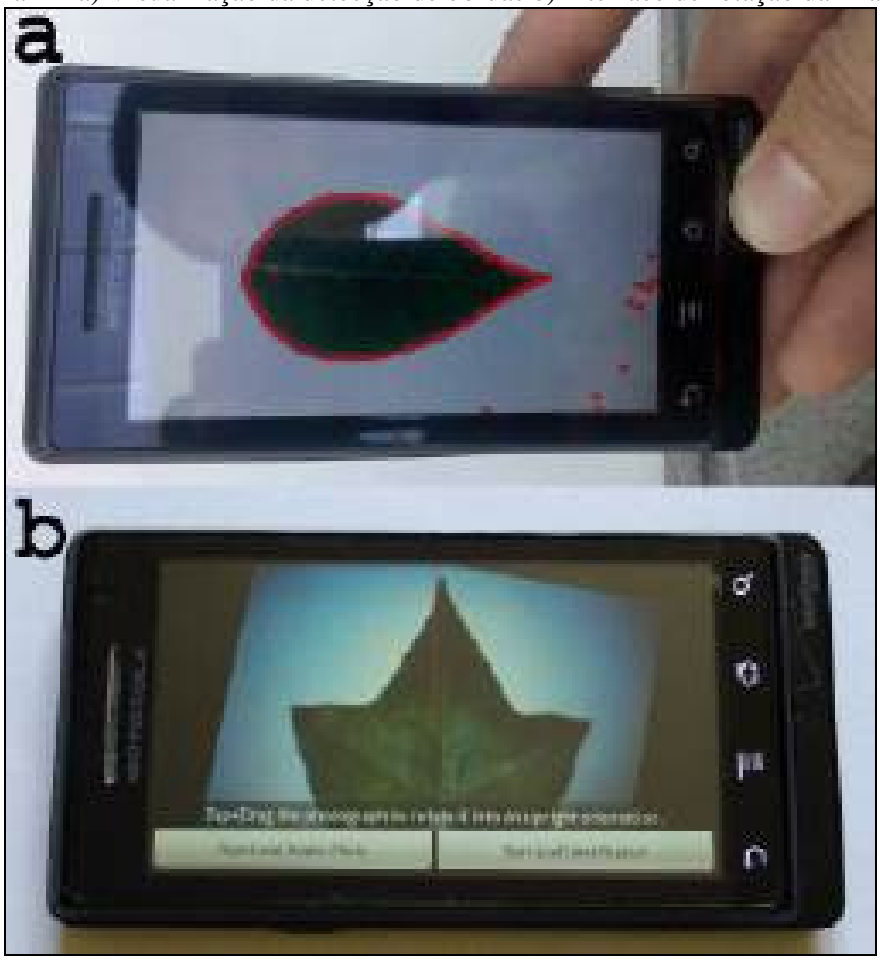
a) aquisição da imagem: a imagem de entrada é adquirida com um smartphone. Esta imagem deve ser capturada em uma distância razoável, com boas condições de luz e em um plano de fundo que ofereça um bom contraste em relação à folha;
b) pré-processamento: nesta etapa é convertida a imagem para tons de cinza. Feito isso, é iniciada a segmentação com a função de descobrir quais pixels fazem parte da folha. Em seguida são fechadas as lacunas na folha e removidos alguns ruídos. Em seguida a coloração da imagem é convertida para binário e o interior da folha é subtraído, deixando uma imagem da borda da folha;
c) extração de características morfológicas: essas são utilizadas para a comparação entre a estrutura da imagem adquirida com as amostras cadastradas na base de

dados.

As características morfológicas utilizadas por Knight, Painter e Potter (2010, p. 6) compreendem proporção de aspecto, retangularidade, proporção de área convexa, proporção de perímetro convexo, circularidade, excentricidade, formato, informação espacial sobre a distribuição da folha em diferentes posições em relação ao eixo vertical e por fim histograma de angulação.

A Figura 4 mostra a visualização do processo de detecção de bordas, e a rotação da imagem a fim de encontrar a ponta e definir um eixo que divida em duas partes a imagem na aplicação desenvolvida por Knight, Painter e Potter (2010).

Figura 4 – a) Visualização da detecção de bordas b) interface de rotação da imagem



Fonte: Knight, Painter e Potter (2010, p. 4).

A aplicação desenvolvida por Knight, Painter e Potter (2010) teve os algoritmos de reconhecimento previamente testados no MATLAB. Após isto, os algoritmos do MATLAB serviram como guia para a implementação da aplicação para a plataforma Android.

As funções do MATLAB *convex hull*, *fill area* e *moments of inertia,* por serem especializadas e de mais alto nível, não estavam disponíveis nas bibliotecas padrão do

Android e nem do Java, portanto, foram implementadas manualmente por Knight, Painter e Potter (2010).

Knight, Painter e Potter (2010) optaram por fazer todas as operações para o reconhecimento diretamente no dispositivo, visando que na falta de internet o aplicativo seja capaz de computar localmente os dados e a partir de uma base local ser possível fazer o reconhecimento das espécies.

A fim de melhorar a usabilidade de sua aplicação, Knight, Painter e Potter (2010) sugerem que sejam feitas melhorias no algoritmo nas etapas onde o usuário interage com a segmentação. A implementação realizada por eles utiliza um grande numero de iterações que deveriam ser evitadas, para diminuir o atraso na atualização da imagem exibida nestas etapas.

2.4.3 Shape based plant leaf classification system using Android

No trabalho de Sangle, Shirsat e Bhosle (2013) é descrito o desenvolvimento de uma aplicação Android que proporciona ao usuário a habilidade de identificar espécies vegetais baseado em fotografias das folhas das plantas tiradas por um dispositivo móvel. A aplicação proposta é composta das seguintes etapas:

a) aquisição da imagem: a imagem de uma folha desconhecida é adquirida pela câmera do dispositivo, assumindo que terá boas condições de luz e distância e possuirá um plano de fundo que forneça o contraste suficiente para a imagem;
b) pré-processamento: antes da extração de características morfológicas deve ser encontrada a borda da folha. Isso é feito convertendo a imagem para escala de cinza e depois separando o fundo da parte desejada por segmentação. Em seguida são fechados os buracos e removidas pequenas regiões. Por fim, a imagem é binarizada e o interior da folha é subtraído, sobrando apenas o contorno da folha;
c) extração de características morfológicas: Após efetuar uma longa pesquisa na literatura e testes com várias combinações, o algoritmo final de Sangle, Shirsat e Bhosle (2013) realiza a combinação das características mais adequadas, segundo os autores.

As características morfológicas que Sangle, Shirsat e Bhosle (2013) utilizaram na extração foram proporção de aspecto, retangularidade, proporção de área convexa, proporção do perímetro convexo, circularidade, excentricidade, formato, informação espacial sobre a distribuição da folha em diferentes posições em relação ao eixo vertical e histograma de angulação.

Figura 5 – Processo de captura da imagem.

Fonte: Sangle, Shirsat e Bhosle (2013, p. 7).

Na aplicação de Sangle, Shirsat e Bhosle (2013, p. 7) é exibida a segmentação em tempo real para permitir ao usuário visualizar os resultados da segmentação e capturar a imagem somente quando a região aceitável da folha for demarcada, como mostrado na Figura 5. Então a aplicação solicita outra entrada do usuário, para que ele rotacione a imagem visando encontrar o topo da folha fotografada. Após isso a aplicação exibe uma animação enquanto faz a extração de características. Por último a aplicação exibe o resultado do cálculo como a Figura 6.

Figura 6 – Resultado do cálculo de distância euclidiana

Fonte: Sangle, Shirsat e Bhosle (2013, p. 7).

Na Figura 6 o menor dos valores é considerado como resultado da combinação entre a entrada e a base de dados.

Sangle, Shirsat e Bhosle (2013) deixam como sugestão para trabalhos futuros a melhoria no desempenho, pois o algoritmo desenvolvido por eles possui muitas iterações que poderiam ser evitadas, o que torna a aplicação lenta e com baixa taxa de atualização na

exibição da imagem da câmera. O pequeno atraso gerado pela baixa taxa de atualização pode dificultar que o usuário aponte com precisão folhas mais estreitas.

2.4.4 Seção comparativa dos trabalhos relacionados

O Quadro 1 apresenta de maneira comparativa algumas características dos trabalhos apresentados nesta seção.

Quadro 1 – Características dos trabalhos relacionados

características / trabalhos relacionados	Kumar et al. (2012)	Knight, Painter e Potter (2010)	Sangle, Shirsat e Bhosle (2013)
Coordenadas de GPS	Sim	Não	Não
Extração das características	Servidor	Dispositivo	Dispositivo
Método de entrada	Câmera do dispositivo móvel	Câmera do dispositivo móvel	Câmera do dispositivo móvel
Método de reconhecimento	Com auxílio do usuário	Automático	Automático
Plataforma mobile	iOS	Android	Android

A partir do Quadro 1 é possível perceber que os trabalhos relacionados possuem funcionalidades bastante semelhantes, e que o método de entrada de dados para as aplicações de classificação de plantas em dispositivos móveis tem sido em todos os casos a câmera do próprio dispositivo. Outra característica a ser levada em conta é que dois destes trabalhos foram desenvolvidos na plataforma Android, uma plataforma que está em ascensão e possui características de custo favoráveis para o desenvolvimento.

A extração de características no trabalho de Kumar et al. (2012) é feita em um servidor e isso é bastante proveitoso para a usabilidade da aplicação pois os processadores presentes em dispositivos móveis não são recomendados para cálculos que possuem complexidade elevada. A técnica de envio ao servidor para extração de características utilizada por Kumar et al. (2012) tem o ponto fraco de ser dependente de conexão com a internet. Knight, Painter e Potter (2010) e Sangle, Shirsart e Bhosle (2013) efetuam os cálculos localmente para não depender da conexão com a internet, porém isto afeta o desempenho da aplicação implicando em sua usabilidade.

A classificação automática de espécies utilizada por Knight, Painter e Potter (2010) e Sangle, Shirsart e Bhosle (2013) possui a vantagem de não ser necessária nenhuma interação com o usuário para a seleção da planta. Kumar et al. (2012) propõe uma maneira de classificação onde o usuário deve escolher dentre uma lista de espécies prováveis qual é a espécie correta.

3 DESENVOLVIMENTO

Neste capítulo são apresentadas as etapas do desenvolvimento da aplicação Plantarum. Na seção 3.1 são enumerados os requisitos principais do projeto desenvolvido. A seção 3.2 apresenta a especificação da aplicação, contendo a arquitetura do projeto cliente e a arquitetura do projeto servidor. A seção 3.3 detalha a implementação das principais técnicas e algoritmos utilizados pela aplicação. Por fim, a seção 3.4 apresenta os testes efetuados para a validação da aplicação e os resultados obtidos.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

Os Requisitos Funcionais (RF) e Requisitos Não Funcionais (RNF) da aplicação Plantarum são:

a) permitir ao usuário acesso a câmera com o intuito de fotografar uma amostra (RF);
b) possuir acesso a informações de localização quando disponível (RF);
c) permitir salvar no dispositivo a imagem, com sua data de captura e suas informações de localização (RF);
d) possuir funcionalidade de envio ao servidor, para classificação e cadastro de amostras (RF);
e) possuir um serviço que utilize a API Plantarum (CASSANIGA, 2012) como mecanismo de cadastro e classificação (RF);
f) exibir os resultados das operações no dispositivo móvel (RF);
g) utilizar a plataforma Microsoft .Net com a linguagem C# para o serviço (RNF);
h) utilizar a plataforma JSE com a linguagem Java para o aplicativo móvel (RNF).

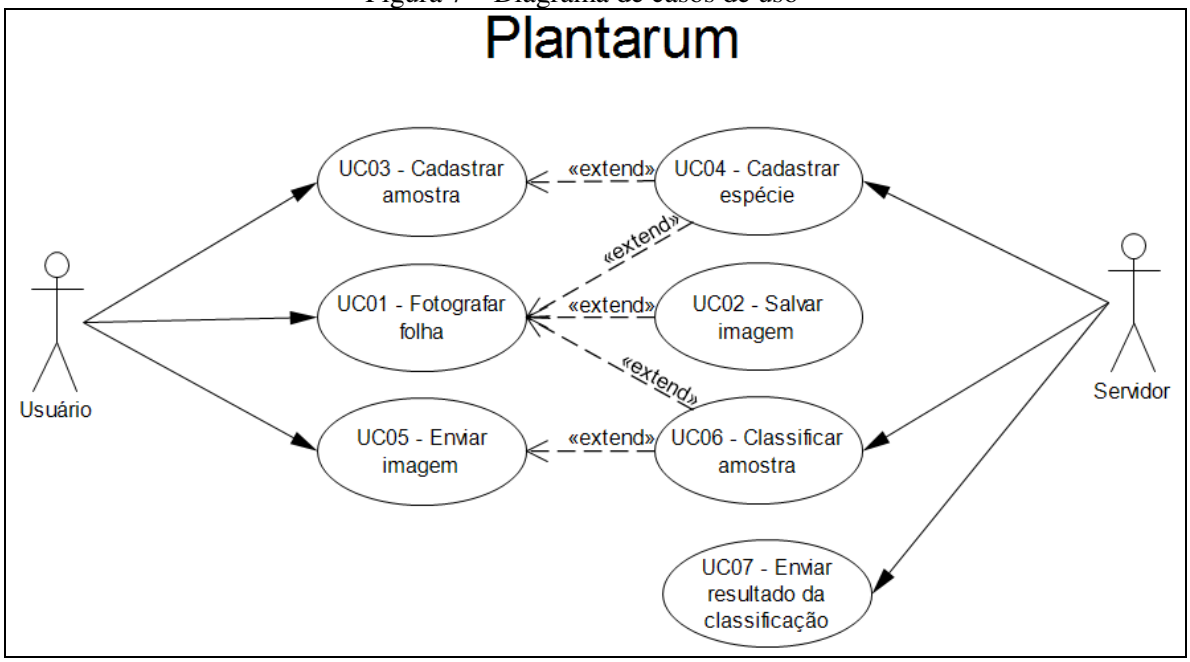
3.2 ESPECIFICAÇÃO

A especificação da aplicação foi representada em diagramas da *Unified Modeling Language* (UML), utilizando a ferramenta ArgoUML 0.34 para descrever a estrutura de utilização da aplicação, a estrutura da aplicação mobile e a jornada dos dados no método principal da API.

3.2.1 Diagrama de casos de uso

A Figura 7 exibe o diagrama de casos de uso com as ações disponibilizadas pela aplicação móvel para o consumo da API Plantarum (CASSANIGA, 2012). Foi identificado apenas um ator, denominado `Usuário`, o qual utiliza todas as funcionalidades da aplicação.

Figura 7 – Diagrama de casos de uso



Segue detalhamento dos casos de uso exibidos no diagrama da Figura 7:

a) `UC01 – Fotografar folha`: permite ao usuário acessar a câmera do dispositivo a fim de que seja fotografada uma amostra;

b) `UC02 – Salvar imagem`: permite que o usuário salve uma amostra no dispositivo móvel na ausência de conexão com o servidor para que seja enviada mais tarde;

c) `UC03 – Cadastrar amostra`: permite que o usuário cadastre uma nova amostra de determinada espécie de planta;

d) `UC04 – Cadastrar espécie`: permite ao usuário adicionar uma nova espécie de planta na base de dados;

e) `UC05 – Enviar imagem`: permite enviar para classificação uma amostra que foi fotografada e salva anteriormente;

f) `UC06 – Classificar amostra`: permite que seja enviada para a classificação uma imagem que acaba de ser fotografada.

g) `UC07 – Enviar resultado da classificação`: permite ao usuário ver o resultado da classificação.

3.2.2 Diagrama de classes

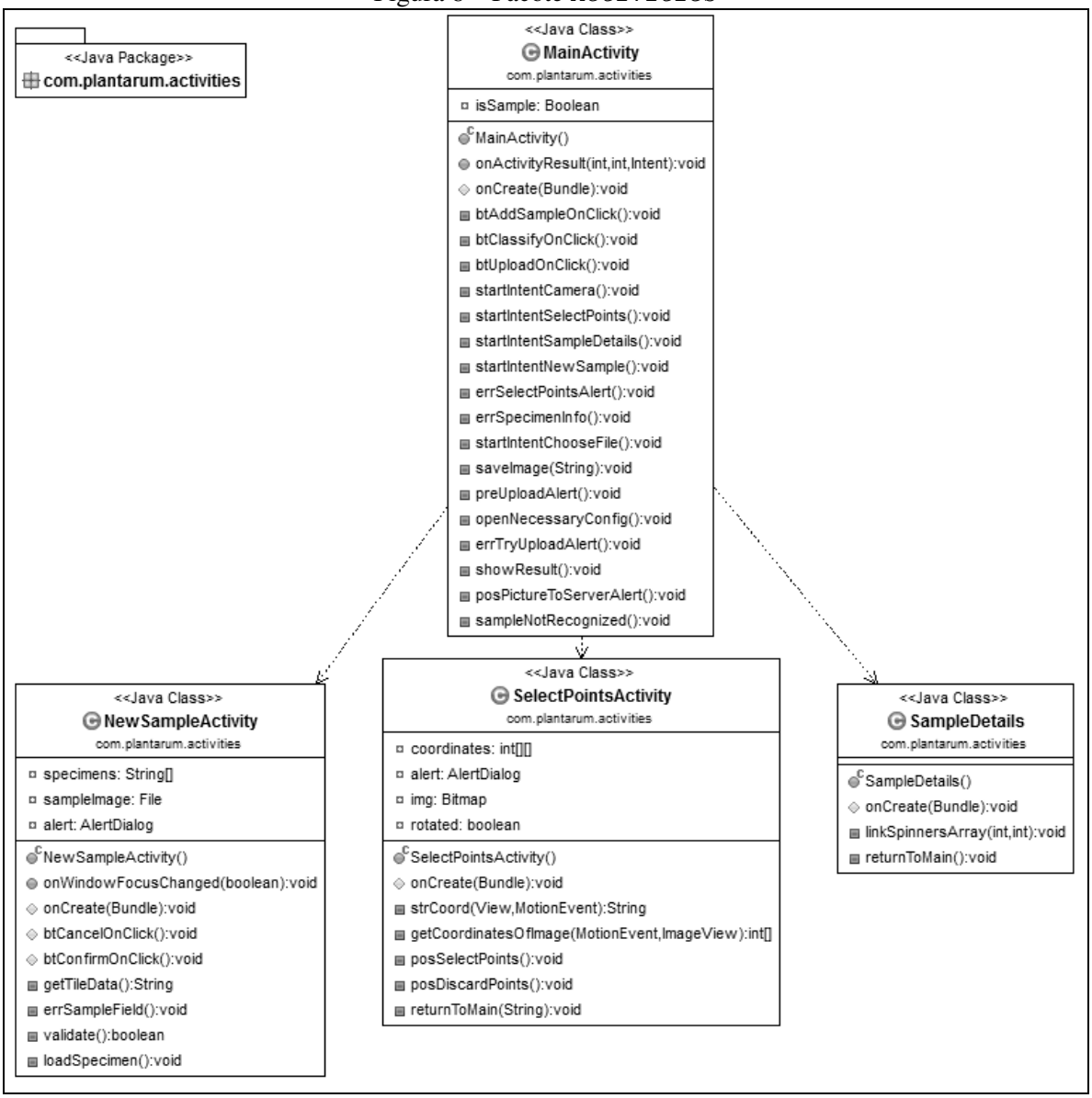
A aplicação é composta por dois projetos diferentes e para facilitar o entendimento, a representação da estrutura das classes foi subdividida em duas subseções. A primeira subseção é referente aos diagramas do aplicativo cliente e a segunda se refere ao diagrama de classes do aplicativo servidor.

3.2.2.1 Diagramas do aplicativo cliente

O aplicativo cliente possui nove classes que foram divididas em dois pacotes, `Activities` e `Utils`.

A Figura 8 apresenta o pacote `Activities` que contem as interfaces acessadas pelo usuário para desempenhar as ações da aplicação.

Figura 8 – Pacote `Activities`



A classe `MainActivity` é a classe que controla a visualização inicial da aplicação e os resultados da classificação de espécies. Esta classe possibilita iniciar a atividade para tirar uma foto através do método `btClassifyOnClick`, cadastrar uma nova amostra com o método `btAddSampleOnClick` ou enviar para o servidor uma imagem previamente salva utilizando o método `btUploadOnClick`.

No método `btClassifyOnClick` o usuário tem acesso à câmera do dispositivo pelo método `startIntentCamera`. Ao retornar da câmera, a aplicação pega as coordenadas de GPS, as adiciona na foto e encaminha o usuário para a tela controlada por `SampleDetails`, onde são solicitadas as informações tipo da folha, pilosidade e discoloridade. No retorno desta atividade o usuário é enviado para a atividade controlada pela classe `SelectPointsActivity` para selecionar os pontos que representam o pecíolo e a ponta da folha.

Após retornar da tela de seleção de pontos, a aplicação salva as coordenadas selecionadas nas metatags da imagem e solicita se o usuário deseja enviar, salvar ou descartar. Ao escolher enviar imagem, esta é enviada ao servidor para a classificação, escolhendo salvar a imagem é salva no dispositivo e escolhendo descartar todas as informações deste processo são descartadas.

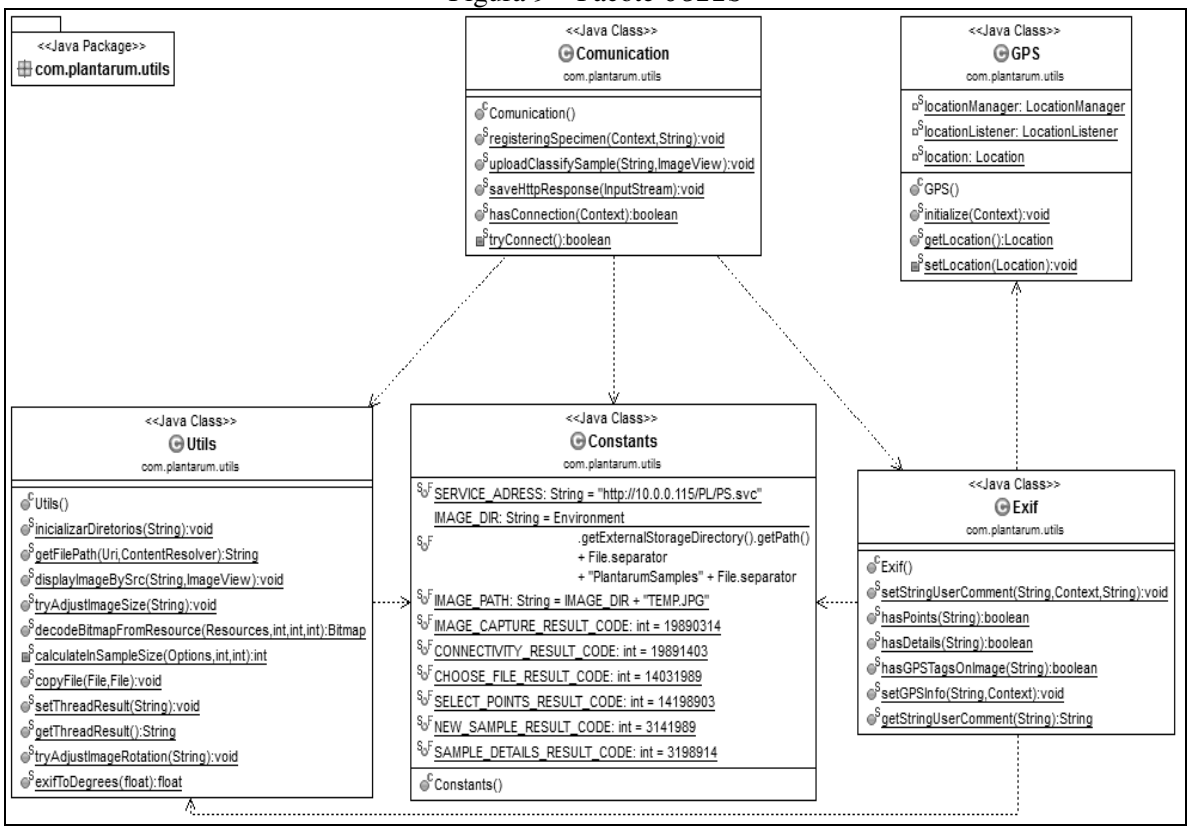
É possível cadastrar uma nova amostra a partir do método `btAddSampleOnClick`. Ao ativar este método a aplicação solicita que o usuário selecione uma imagem que foi salva previamente. Após isto é verificado se a imagem selecionada já possui as informações tipo de folha, pilosidade e discoloridade, e, se não possuir, é solicitado o preenchimento destas informações. Então verifica-se a imagem para encontrar as informações de pecíolo e ponta da folha e se não forem encontradas o usuário deve selecionar na imagem onde estão estas coordenadas.

Por fim é iniciada uma nova atividade controlada pela classe `NewSampleActivity`, onde o usuário deve preencher a informação de nome da amostra, uma descrição e se esta amostra representa a espécie. No fim do processo o usuário pode confirmar ou cancelar a inclusão da amostra. Ao confirmar, a aplicação salva as informações na imagem e envia a mesma para o servidor utilizando o método `registeringSpecimen`.

A aplicação permite também que seja enviada para classificação uma imagem salva no dispositivo. Para isto é utilizado o método `btUploadOnClick`, que solicita a seleção de uma imagem e então são feitas as mesmas verificações das informações da imagem feitas no processo de cadastrar amostra. Por fim a imagem é enviada para a classificação no servidor. Se a imagem não puder ser classificada, é questionado ao usuário se ele deseja cadastrar uma nova espécie com esta imagem, e se ele concordar é iniciado o processo de cadastro de amostra.

Para a realização das tarefas desempenhadas são necessários alguns métodos de apoio, por isso foi criado o pacote `utils` que contém funcionalidades comuns entre as tarefas. A Figura 9 expõe o pacote utils e cada uma de suas classes.

Figura 9 – Pacote `Utils`

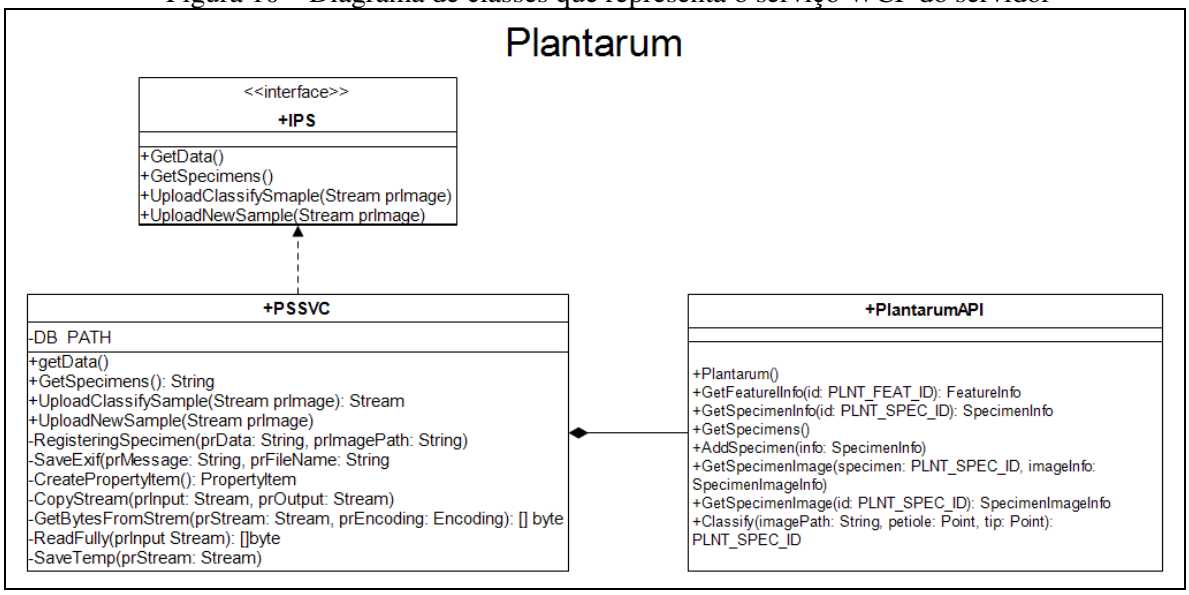


O pacote `utils` é composto pelas classes `Utils`, `Comunication`, `GPS`, `Constants` e `Exif`. A classe `Utils` contém métodos diversos para o tratamento de arquivos, a exibição das imagens em seu tamanho correto no dispositivo e para tratar o resultado da comunicação assíncrona com o servidor. A classe `Constants` contém constantes comuns para o uso entre as atividades. A classe `GPS` encapsula o acesso ao GPS do dispositivo, provendo informações de localização. A classe `Exif` gerencia a aquisição e o armazenamento de metatags em arquivos. Por fim, `Comunication` gerencia as transferências de dados do aplicativo cliente para o servidor.

3.2.2.2 Diagrama de classes do aplicativo servidor

As classes envolvidas em responder as requisições enviadas pela aplicação cliente estão representadas pela Figura 10.

Figura 10 – Diagrama de classes que representa o serviço WCF do servidor



A interface IPS contém os métodos que são possíveis acessar de fora do servidor. O método `GetData` retorna um texto simples confirmando que a conexão com o servidor foi efetuada com sucesso, e que o serviço está executando corretamente. O método `GetSpecimens` utiliza a classe `PlantarumAPI` para retornar uma lista com todas as espécies da base em que a aplicação for se conectar.

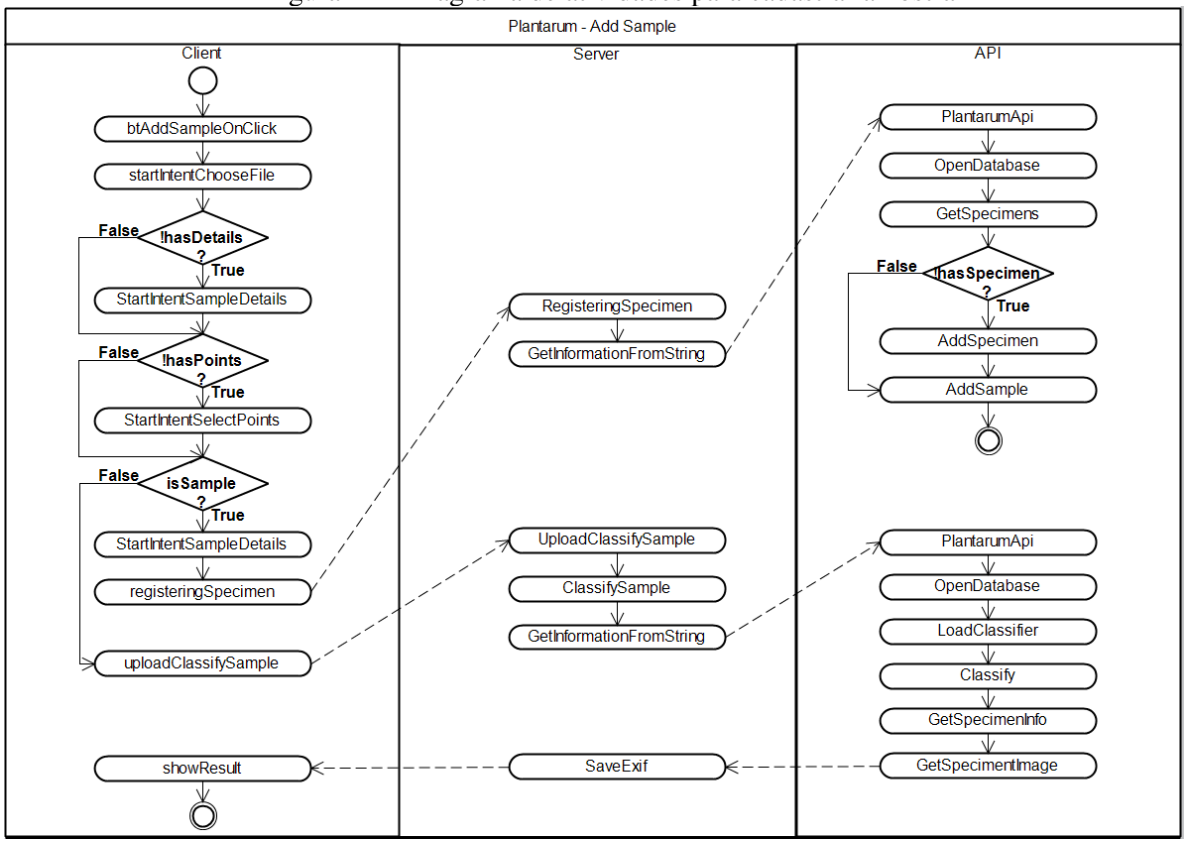
Quando o cliente chamar o método `UploadClassifySpecimen`, o servidor irá receber os dados do cliente e chamar o método `ClassifySpecimen`. O retorno deste método será a imagem e o nome da espécie que foi reconhecida, então o servidor agrupa essas informações e retorna para o cliente.

O cliente também pode chamar o método `UploadNewSample`, que internamente fará a recepção dos dados e chamará o `RegisteringNewSample` para fazer o cadastro da amostra e também da espécie, se esta ainda não existir.

3.2.3 Diagramas de atividades

Os diagramas de atividades apresentados a seguir nos ajudam a visualizar a sequência das operações executadas no cliente e no servidor nas duas funções mais complexas da aplicação. A Figura 11 mostra o diagrama de sequência para a função de cadastrar amostra.

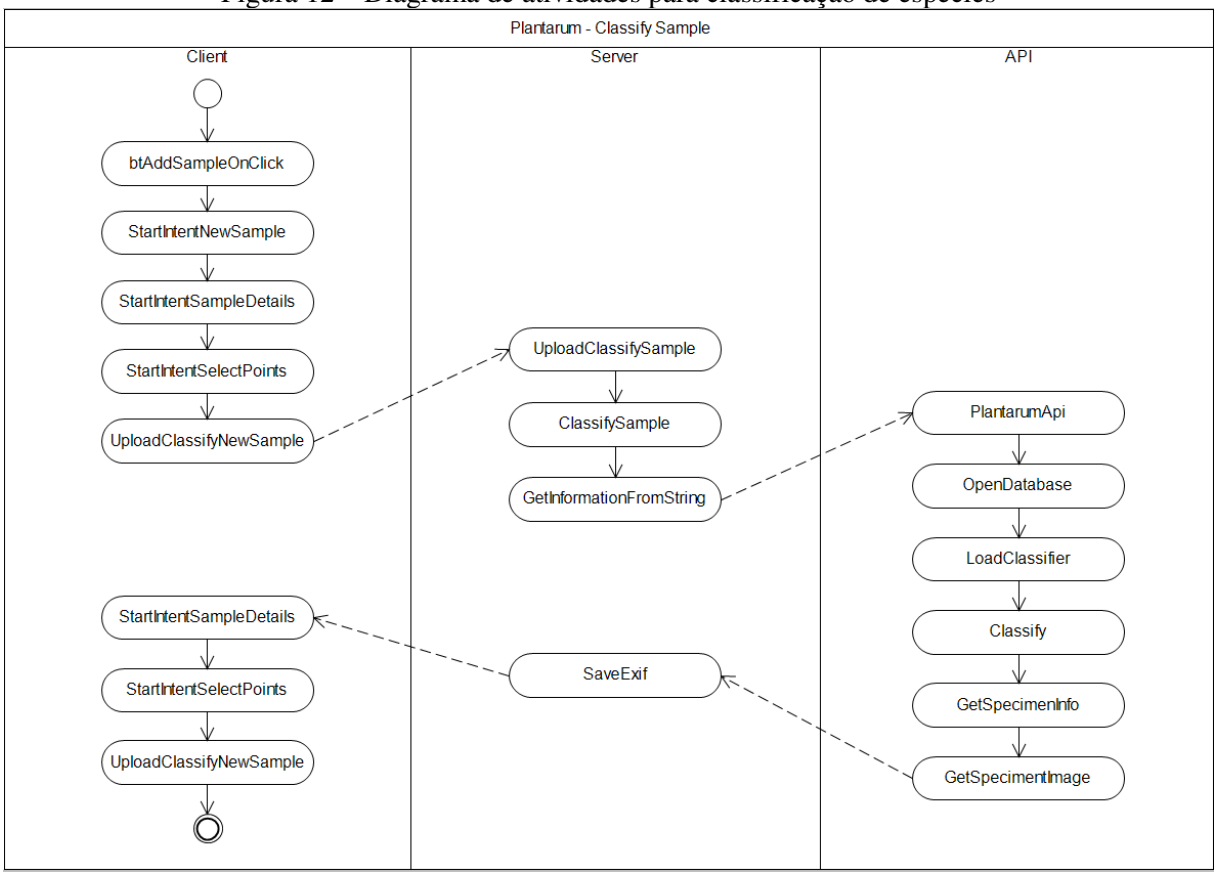
Figura 11 – Diagrama de atividades para cadastrar amostra



Na Figura 11 mostra a atividade de cadastrar amostra na situação mais descritiva, onde todos os casos de uso podem ser executados dependendo do contexto que o usuário estiver. Se o usuário estiver retornando, entrando no cadastro de amostra após o retorno do método classificar espécie e não tiver sido possível classificar a imagem, o usuário já terá uma amostra com as coordenadas de pecíolo e ponta cadastradas, então as operações de criação `startIntentChooseFile`, `startIntentSampleDetails` e `startIntentSelectPoints` não serão executadas. Também se o usuário ao digitar o nome da espécie, selecionar um nome já existente, a atividade de criação `AddSpecimen` não será executada.

A Figura 12 mostra o diagrama de atividades para o processo de classificação de especies.

Figura 12 – Diagrama de atividades para classificação de espécies



Da mesma forma que a Figura 11, a Figura 12 mostra o processo que se refere da maneira mais descritiva possível, podendo não executar dois processos de criação dependendo do contexto do usuário.

Se a imagem capturada pelo usuário não conseguir ser classificada junto a base de dados, os processos de criação `GetSpecimenInfo` e `GetSpecimenImage` não serão invocados.

Mais detalhes de como os métodos funcionam poderão ser vistos na seção de implementação.

3.3 IMPLEMENTAÇÃO

A seguir são mostradas as técnicas e ferramentas utilizadas e a operacionalidade da implementação.

3.3.1 Técnicas e ferramentas utilizadas

A aplicação do servidor foi desenvolvida utilizando a linguagem C#, disponibilizada pela plataforma Microsoft .Net. Como ambiente de desenvolvimento foi selecionada a IDE Microsoft Visual Studio 2012. Neste projeto foi utilizado como tecnologia a API Plantarum (CASSANIGA, 2012) para a classificação e cadastro de espécies.

Para melhorar os resultados deste trabalho foram incluídas novas características para a classificação e reconhecimento de plantas na API Plantarum de Cassaniga (2012). Para isto foi necessário utilizar o mesmo conjunto de ferramentas e técnicas utilizadas para a construção desta API.

Na aplicação do cliente foi utilizada a linguagem Java, oferecida pela Oracle. Como plataforma de desenvolvimento foi escolhida a IDE Android Developer Tools (ADT) versão 22.3.0, uma customização da IDE Eclipse oferecida pela Google para o desenvolvimento para dispositivos Android, em complemento foi utilizado o SDK versão 22.3. Para a comunicação via HTTP foram utilizadas as bibliotecas Httpclient-4.3, Httpcore-4.3 e Httpmime-4.3, disponibilizadas pela Apache Software Foundation. Para a leitura e gravação de propriedades dos arquivos de imagem, foi utilizada a biblioteca independente Exifdriver.

3.3.1.1 Implementação da aplicação cliente

A aplicação cliente foi desenvolvida com base no dispositivo Samsung Galaxy S IV com o Android na versão mínima de 2.3.3. Este dispositivo possui processador quad-core de 1.2 GHz, 2 GB de memória, câmera de 13 megapixels e resolução da tela de 1080x1920 pixels. A aplicação exige permissões de internet, acessar e modificar o estado da rede sem fio, acessar localização global, acessar a câmera, escrita e leitura de arquivos.

A aplicação inicia na `MainActivity`, que disponibiliza três ações ao usuário, capturar imagem, adicionar amostra e fazer classificação de amostra existente. A captura da imagem é iniciada quando o usuário clica no botão `btCamera`. A ação do botão inicia uma nova `Intent` que o direciona para câmera permitindo que ele ajuste as configurações para a foto manualmente e fotografe a amostra. O método que inicia a câmera é o `startIntentCamera` e sua implementação pode ser vista no Quadro 2.

Quadro 2 – Iniciando a `Intent` para acessar a câmera

```
 1  private void startIntentCamera() {
 2    try {
 3      Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
 4      Uri outputFileUri = Uri.fromFile(new File(Constants.IMAGE_PATH));
 5      intent.putExtra(MediaStore.EXTRA_OUTPUT, outputFileUri);
 6      startActivityForResult(intent,
 7                             Constants.IMAGE_CAPTURE_RESULT_CODE);
 8
 9    } catch (IllegalArgumentException e) {
10      @SuppressWarnings("unused")
11      String tem = e.getMessage() + ";" + e.getLocalizedMessage() + ";"
12                  + e.fillInStackTrace();
13    }
14  }
```

Ao iniciar a câmera é necessário passar um caminho para o arquivo que será gerado e como é necessário fazer operações, também é passado um valor para o retorno da `Intent`, para assim ser possível tratar de maneira independente cada atividade iniciada.

Após ser executada a `Intent`, a aplicação retorna para o método `onActivityResult`, e pelo código de início da `Intent` é possível saber que o retorno é da câmera. A partir deste ponto a aplicação faz conexão com GPS e atribui os valores de latitude e longitude para a imagem.

Para conseguir as informações de GPS é necessário criar um `LocationListner`, assim sempre que houver atualizações em um objeto do tipo `Location`, as coordenadas de localização serão atualizadas. Por este motivo foi implementado a classe `GPS` a fim de manter um objeto do tipo `Location` sempre atualizado para que quando for necessário utilizar as coordenadas elas estejam corretas e atualizadas. O Quadro 3 contém a implementação da classe `GPS`.

Quadro 3 – Classe `GPS`

```
 1  package com.plantarum.utils;
 2
 3  import android.content.Context;
 4  import android.location.Location;
 5  import android.location.LocationListener;
 6  import android.location.LocationManager;
 7  import android.os.Bundle;
 8
 9  public class GPS {
10
11    private static LocationManager locationManager;
12
13    private static LocationListener locationListener;
14
15    private static Location location;
16
17    public static void initialize(Context context) {
18      setLocation(null);
19      locationListener = new LocationListener() {
20        public void onLocationChanged(Location loc) {
21          GPS.setLocation(loc);
22        }
23
24        public void onStatusChanged(String provider,
25                                    int status, Bundle extras) {
26        }
27
28        public void onProviderDisabled(String provider) {
29        }
30
31        public void onProviderEnabled(String provider) {
32        }
33      };
34
35      locationManager = (LocationManager) context
36          .getSystemService(Context.LOCATION_SERVICE);
37      locationManager.requestLocationUpdates(
38                                    LocationManager.GPS_PROVIDER,
39                                    5000, 0, locationListener);
40    }
41
42    public static Location getLocation() {
43      return location;
44    }
45
46    private static void setLocation(Location location) {
47      GPS.location = location;
48    }
49  }
```

Para atribuir os valores de `GPS` para a imagem, a aplicação faz uma chamada ao método `setGPSInfo` logo após a confirmação da imagem fotografada, então neste método as coordenadas são adquiridas e salvas nas propriedades EXIF específicas para localização. O Quadro 4 contém a implementação do método `setGPSInfo`.

Quadro 4 – Implementação do método `setGPSInfo`

```
 1    public static void setGPSInfo(String prImgPath, Context context) {
 2      try {
 3        File sampleImage = new File(prImgPath);
 4        Location loc = GPS.getLocation();
 5
 6        if (loc == null)
 7          GPS.initialize(context);
 8
 9        for (int i = 0; ((i < 10) && (loc == null)); i++) {
10          loc = GPS.getLocation();
11        }
12
13        ExifManager exifManager;
14        ExifDriver exifDriver = ExifDriver.getInstance(sampleImage
15            .getAbsolutePath());
16        if (exifDriver != null) {
17          exifManager = new ExifManager(exifDriver, context);
18          exifManager.setGPSLocation(loc.getLatitude(),
19              loc.getLongitude(), loc.getAltitude());
20
21          exifDriver.save(sampleImage.getParent() + File.separator
22              + ".temp2.jpg");
23        }
24        Utils.copyFile(new File(sampleImage.getParent() + File.separator
25            + ".temp2.jpg"), sampleImage);
26
27        new File(sampleImage.getParent() + File.separator +
28                                          ".temp2.jpg")
29                                                      .delete();
30
31        sampleImage = new File(sampleImage.getAbsolutePath());
32
33      } catch (Exception e) {
34        Toast.makeText(
35            context,
36            "Não foi possível salvar os dados de localização na imagem",
37            Toast.LENGTH_LONG).show();
38      }
39    }
```

Após a operação de salvar as informações de localização na imagem, o usuário é direcionado para uma nova tela a fim de que sejam selecionados o pecíolo e a ponta da folha. Esta nova tela é controlada pela classe `SelectPointsActivity`. Os pontos a serem registrados devem ser marcados pelo usuário na ordem correta, primeiro pecíolo e depois ponta.

Para marcar os pontos na folha o usuário clica na tela para apontar onde fica o pecíolo e a ponta da amostra, porém como a imagem não necessariamente será da mesma altura e largura da tela que a está exibindo, a imagem é transformada para ocupar a tela toda. Então quando o usuário clica em algum ponto na tela a fim de marcar a imagem, é necessário que se

faça o cálculo de proporção das dimensões da matriz da imagem com as proporções da matriz da tela do dispositivo, para que sejam transformadas as coordenadas do clique também.

As coordenadas obtidas com o clique do usuário estão relacionadas às dimensões da tela e os cálculos para a transformação do clique do usuário foram implementados no método `getCoordinatesOfImage`, onde é recebido o parâmetro o evento do click `MotionEvent` e a componente `ImageView` que hospeda a imagem na tela. Este procedimento serve para obter as coordenadas de dispositivo normalizado NDC. A implementação deste método pode ser vista no Quadro 5.

Quadro 5 – Convertendo coordenadas da tela para coordenadas da imagem

```
 1  private int[] getCoordinatesOfImage(MotionEvent event, ImageView view)
 2  {
 3        float eventX = event.getX();
 4        float eventY = event.getY();
 5        float[] eventXY = new float[] { eventX, eventY };
 6
 7        Matrix invertMatrix = new Matrix();
 8        view.getImageMatrix().invert(invertMatrix);
 9
10        invertMatrix.mapPoints(eventXY);
11        int x = Integer.valueOf((int) eventXY[0]);
12        int y = Integer.valueOf((int) eventXY[1]);
13
14        Drawable imgDrawable = view.getDrawable();
15        Bitmap bitmap = ((BitmapDrawable) imgDrawable).getBitmap();
16
17        if (x < 0) {
18              x = 0;
19        } else if (x > bitmap.getWidth() - 1) {
20              x = bitmap.getWidth() - 1;
21        }
22
23        if (y < 0) {
24              y = 0;
25        } else if (y > bitmap.getHeight() - 1) {
26              y = bitmap.getHeight() - 1;
27        }
28
29        int result[] = new int[2];
30        result[0] = y;
31        result[1] = x;
32
33        return result;
34  }
```

O método no `getCoordinatesOfImage` é chamado toda vez que o usuário toca uma região no `ImageView` que mostra a imagem, pois foi adicionado um `onTouchListener` neste primeiro, e no evento `onTouch` foi adicionada a chamada para o método `getCoordinatesOfImage`. Ao clicar pela segunda vez, a aplicação assume que foram selecionadas as duas coordenadas necessárias para a imagem, então é exibido um

`AlertDialog` perguntando se o usuário deseja manter, inverter ou descartar os pontos. Ao descartar, a aplicação espera que ele selecione novamente as coordenadas. Ao manter ou inverter ele volta para a tela inicial e os pontos são salvos nas propriedades EXIF da imagem.

Para salvar as coordenadas na imagem é utilizado o método `setStringUserComment`, que recebe como parametro uma `String` que será salva na propriedade `UserComment` das propriedades EXIF da imagem, pois esta propriedade pode receber um texto livre. Fez-se necessário a adição da biblioteca Exifdriver pois as bibliotecas nativas do Java não possuem tratamento para a propriedade necessária e as bibliotecas padrão do Android possuem um número limitado de opções que podem ser utilizadas, sendo que nenhuma delas permitia gravação de texto livre. O Quadro 6 mostra a implementação do método `setStringUserComment`.

Quadro 6 – Salvando informações na imagem

```
 1    public static void setStringUserComment(final String comment,
 2                                            final Context context,
 3                                                 final String prImgPath)
 4  {
 5      ExifManager exifManager;
 6      String nomeArquivo = Constants.IMAGE_DIR + ".tempS.jpg";
 7      String s = "";
 8      File sampleImage = new File(prImgPath);
 9
10      Utils.copyFile(sampleImage, new File(nomeArquivo));
11
12      sampleImage = new File(nomeArquivo);
13      try {
14        ExifDriver exifDriver = ExifDriver.getInstance(prImgPath);
15        if (exifDriver != null) {
16          if (exifDriver.getIfd0().get(270) != null) {
17            byte[] temp = new byte[exifDriver.getIfd0().get(270)
18                                              .getBytes().length];
19            temp = exifDriver.getIfd0().get(270).getBytes();
20
21            s = EncodingUtils.getAsciiString(temp);
22          }
23
24          s = s + ";#;" + comment;
25
26        exifManager = new ExifManager(exifDriver, context);
27        exifManager.setImageDescription(s);
28        exifDriver.save(Constants.IMAGE_DIR + ".temp2.jpg");
29
30        exifDriver = ExifDriver.getInstance(Constants.IMAGE_DIR
31                                            + ".temp2.jpg");
32        if ((exifDriver != null) &&
33            (exifDriver.getIfd0() != null) &&
34            (exifDriver.getIfd0().get(270) != null)){
35          String temp = EncodingUtils.getAsciiString(
36                                           exifDriver.getIfd0()
37                                          .get(270).getBytes());
38
39          if (temp.contains(s))
41            Utils.copyFile(new File(Constants.IMAGE_DIR
42                                    + ".temp2.jpg"),
43                                     new File(Constants.IMAGE_PATH));
44          else
45            Utils.copyFile(new File(prImgPath),
46                           new File(Constants.IMAGE_PATH));
47        }
48
49        new File(Constants.IMAGE_DIR + ".temp2.jpg").delete();
50        new File(Constants.IMAGE_DIR + ".tempS.jpg").delete();
51      }
52        sampleImage = null;
53      } catch (Exception e) {
54        e.getMessage();
55      }
56    }
```

Após a chamada do método `setStringUserComent` é exibido um `AlertDialog` para que o usuário selecione se deseja enviar, armazenar ou descartar a imagem. Ao selecionar descartar, o usuário volta para a tela inicial sem ação nenhuma salva. Ao selecionar armazenar, a imagem é salva no dispositivo para que o usuário seja capaz de mais tarde selecioná-la e enviar a imagem para classificação ou para cadastrá-la como nova espécie. O método utilizado para salvar a imagem é `saveImage` e o aplicativo cria um novo arquivo em um diretório do armazenamento interno com o nome correspondendo a data e hora do dispositivo no momento de salvar. O Quadro 7 contém a implementação deste método.

Quadro 7 – Salvando a imagem no dispositivo

```
 1    private void saveImage(String prSrc) {
 2      try {
 3        File src = new File(prSrc);
 4        SimpleDateFormat sdf = new SimpleDateFormat("yyyyMMddHHmmss");
 5
 6        File f = new File(Constants.IMAGE_DIR
 7            + "SampleDate" + sdf.format(new Date()) + ".jpg");
 8
 9        Utils.copyFile(src, f);
10
11        Toast.makeText(MainActivity.this,
12                       "Imagem armazenada com sucesso.",
13                       Toast.LENGTH_SHORT).show();
14      } catch (Exception e) {
15        Toast.makeText(this, "Erro ao salvar arquivo",
16                                                  Toast.LENGTH_LONG)
17            .show();
18      }
19    }
```

Após salvar a imagem o usuário retorna à tela inicial e pode fotografar uma nova imagem, cadastrar uma amostra ou enviar uma amostra para classificação.

No `AlertDialog` que é exibido após a chamada do método `setStringUserComent`, se o usuário clicar em enviar a aplicação verificará se possui conexão com a internet, e, se não possuir é exibido um `AlertDialog` informando que é necessária conexão com redes de dados, pedindo ao usuário para ativar as configurações. Ao recusar ele é direcionado para a tela inicial e se ele aceitar o aplicativo faz uma chamada ao método `openNecessaryConfig` que encaminha o usuário para as configurações de internet e localização. O Quadro 8 demonstra como ficou a implementação do método `openNecessaryConfig`.

Quadro 8 – Salvando a imagem no dispositivo

```
 1    private void openNecessaryConfig() {
 2      WifiManager wifiManager = (WifiManager)
 3                                getSystemService(WIFI_SERVICE);
 4      if (!wifiManager.isWifiEnabled())
 5        startActivityForResult(
 6                          (new Intent(Settings.ACTION_WIFI_SETTINGS)),
 7                                  Constants.CONNECTIVITY_RESULT_CODE);
 8
 9      Location loc = GPS.getLocation();
10      if (loc == null)
11        startActivityForResult((new Intent(
12            Settings.ACTION_LOCATION_SOURCE_SETTINGS)),
13            Constants.CONNECTIVITY_RESULT_CODE);
14    }
```

O método `openNecessariConfig` verifica se a rede *wireless* está habilitada e se não estiver inicia uma nova `Intent` que leva às configurações de dados para que o usuário ative e conecte em uma rede. Também tenta conseguir conexão com o GPS e se não conseguir inicia uma nova `Intent` que leva às configurações de GPS para que o usuário o ative, pois caso a imagem ainda não possua as informações de localização ele tentará adicionar novamente antes de enviar ao servidor.

Após isso, se o usuário estiver com as configurações de rede e GPS ativas, a aplicação tenta enviar a imagem para classificação pelo método `uploadClassifySample`. Neste método a aplicação tenta conectar-se ao servidor para enviar a imagem. A plataforma Android requer que todas as requisições assíncronas sejam feitas em uma *thread* separada da atividade principal, para garantir o funcionamento da aplicação e forçar o desenvolvedor a controlar o tempo máximo para obter uma resposta do servidor. O Quadro 9 mostra como ficou o código deste método.

Quadro 9 – Enviando a imagem para classificação

```
 1  public static void uploadClassifySample(final String prImgPath,
 2                                          final ImageView ivImage) {
 3    Thread t = new Thread(new Runnable() {
 4      public void run() {
 5        try {
 6          HttpParams httpParameters = new BasicHttpParams();
 7          HttpConnectionParams.setConnectionTimeout(httpParameters,
 8                                                    999999999);
 9          HttpConnectionParams.setSoTimeout(httpParameters, 99999999);
10          HttpClient httpClient = new DefaultHttpClient(httpParameters);
11          HttpPost httppost = new HttpPost(Constants.SERVICE_ADRESS
12                                       + "/UploadClassifySample");
13          FileBody fb = new FileBody(new File(prImgPath),
14                                ContentType.APPLICATION_JSON);
15
16          MultipartEntityBuilder multipartEntity =
17                                              MultipartEntityBuilder
18                                                          .create();
19          multipartEntity.setMode(HttpMultipartMode.STRICT);
20          multipartEntity.addPart("image/jpeg", fb);
21          httppost.setEntity(multipartEntity.build());
22
23          HttpResponse httpResp = httpClient.execute(httppost);
24
25          HttpEntity entityResponse = httpResp.getEntity();
26          InputStream is = entityResponse.getContent();
27          if (httpResp.getStatusLine().getStatusCode() != 200)
28            throw new Exception();
29
30          salvarRetornoServidor(is);
31
32          String s = Exif.getStringUserComment(Constants.IMAGE_DIR
33                                                    + "RetServ.jpg");
34          if (s.contains("specInfo")){
35            Utils.setThreadResult("OK");
16            return;
37          }
38          if (s.contains("NotRecognized")){
39            Utils.setThreadResult("NotRecognized");
40            return;
41          }
42          if (s.contains("ERR")){
43            Utils.setThreadResult("ERR");
44            return;
45          }
46        } catch (Exception e) {
47          Utils.setThreadResult("ERR");
48        }
49      }
50    });
51    t.start();
52    try {
53      t.join();
54    } catch (InterruptedException e) {
55      e.printStackTrace();
56    }
57  }
```

O resultado do servidor para a classificação será a imagem que representa a espécie da amostra que foi submetida e o nome da espécie estará salvo na propriedade EXIF do arquivo enviado. Este resultado será salvo em um arquivo para que, ao retornar para a atividade principal, seja exibida a imagem e a espécie reconhecida. Caso a amostra não for reconhecida a aplicação salva um arquivo contendo a resposta do servidor e mostra um `AlertDialog` que informa ao usuário que a amostra não foi encontrada. Em seguida pergunta se ele deseja cadastrar uma nova espécie com base na imagem enviada.

Para exibir o resultado no dispositivo é necessário ler o `Stream` recebido e salvá-lo em um arquivo. Após isso é chamado o método `showResult` que lê este arquivo, o decodifica como bitmap e adquire a informação que está nas propriedades EXIF para mostrar o nome da espécie. O Quadro 10 contém a implementação necessária para exibir o resultado.

Quadro 10 – Exibindo o resultado da classificação

```
 1  private void showResult() {
 2    Bitmap bMap = BitmapFactory.decodeFile(sampleImage
 3        .getAbsolutePath());
 4
 5    if (bMap.getWidth() > bMap.getHeight()) {
 6      Matrix matrix = new Matrix();
 7      matrix.postRotate(90);
 8      bMap = Bitmap.createBitmap(bMap, 0, 0, bMap.getWidth(),
 9          bMap.getHeight(), matrix, true);
10    }
11
12    ((ImageView) findViewById(R.id.ivImagem)).setImageBitmap(bMap);
13
14    String s = "";
15    s = getStringUserComment(sampleImage.getAbsolutePath());
16
17    ((TextView) findViewById(R.id.tvResultado)).setText(s);
18  }
```

Ao fim será mostrada na tela inicial a imagem que representa a espécie e o nome da espécie que foi reconhecida.

3.3.1.2 Implementação da aplicação do servidor

A aplicação do servidor foi desenvolvida no modelo WCF para permitir o consumo de seu serviço no ambiente da internet e o *framework* do .Net utilizado foi o 4.0. Para a classificação e cadastro de imagens foi utilizada a API Plantarum (CASSANIGA, 2012) e suas dependências. A aplicação não faz uso de nenhuma biblioteca além da mencionada acima.

O modelo WCF é formado por um contrato que fica público e contém a assinatura para a utilização dos serviços implementados. O contrato da aplicação desenvolvida possui quatro

serviços para o consumo público. O Quadro 11 demonstra a estrutura do contrato implementado.

Quadro 11 – Contrato da aplicação do servidor

```
 1  using System.IO;
 2  using System.ServiceModel;
 3  using System.ServiceModel.Web;
 4
 5  namespace PL
 6  {
 7      [ServiceContract]
 8      public interface IPS
 9      {
10          [OperationContract]
11          [WebGet(UriTemplate = "/GetData",
12              RequestFormat   = WebMessageFormat.Json,
13              ResponseFormat  = WebMessageFormat.Json)]
14          string GetData();
15
16          [OperationContract]
17          [WebGet(  BodyStyle = WebMessageBodyStyle.Bare,
18                  UriTemplate = "/GetSpecimens",
19               ResponseFormat = WebMessageFormat.Xml,
20                RequestFormat = WebMessageFormat.Json)]
21          string GetSpecimens();
22
23          [OperationContract]
24          [WebInvoke(Method = "POST",
25                  BodyStyle = WebMessageBodyStyle.Bare,
26                 UriTemplate = "/UploadClassifySample",
27              RequestFormat = WebMessageFormat.Json,
28             ResponseFormat = WebMessageFormat.Xml)]
29          Stream UploadClassifySample(Stream imagem);
30
31          [OperationContract]
32          [WebInvoke(Method = "POST",
33                  BodyStyle = WebMessageBodyStyle.Bare,
34                UriTemplate = "/UploadNewSample",
35              RequestFormat = WebMessageFormat.Json,
36             ResponseFormat = WebMessageFormat.Xml)]
37          string UploadNewSample(Stream imagem);
38      }
39  }
```

No contrato exibido no Quadro 11 existem quatro métodos para atender a aplicação cliente desenvolvida. São dois métodos do tipo `GET`, sem parâmetros, com o formato da requisição e da resposta em `Json`. Em complemento, mais dois métodos do tipo `POST` que recebem uma imagem no formato `Stream` como parâmetro, com o formato da requisição em `Json` e o formato da resposta em `XML`.

O método `GetData` somente retorna uma mensagem simples do servidor, é utilizado para testar se a conexão com o servidor foi efetuada com sucesso e se o serviço está executando de maneira adequada.

O método `GetSpecimens` retorna o nome de todas as espécies cadastradas na base de dados na forma textual, separadas por um conjunto de caracteres específico utilizado para dividir o texto simples em uma lista com todos os nomes. O Quadro 12 contém o código utilizado neste método.

Quadro 12 – Retornando os nomes das espécies da base de dados

```
 1  String IPS.GetSpecimens()
 2  {
 3              string result = "";
 4
 5               api = new PlantarumAPI();
 6               api.OpenDatabase(DB_PATH);
 7               var especies = api.GetSpecimens();
 8
 9               foreach (var esp in especies)
10               {
11                   result += esp.Name + ";%;";
12               }
13
14               if (string.IsNullOrEmpty(result))
15               {
16                   result = "N/A;%;";
17               }
18
19               return result;
20  }
```

Os métodos do servidor utilizam uma base de dados que seu caminho fica definida na constante `DB_PATH`. No método `GetSpecimens` a aplicação do servidor cria um objeto novo que inicializa a API, em seguida abre a base de dados e chama o método que retorna todas as espécies cadastradas. Após isso percorre a lista retornada para concatenar em uma variável de texto os nomes destas espécies com o conjunto de caracteres de separação e por fim retorna o texto.

O método `UploadClassifySample` é utilizado para classificar uma amostra que é recebida por parâmetro, nesta imagem deve conter os pontos do pecíolo e da ponta da folha nas propriedades EXIF da imagem. O Quadro 13 mostra a implementação deste método.

Quadro 13 – Recebendo uma imagem para classificação

```
 1  Stream IPS.UploadClassifySample(Stream prImage)
 2  {
 3    try
 4    {
 5      var fileName = string.Format("{0}.jpg",
 6                           DateTime.Now.ToString("yyyyMMddHHmmss"));
 7      fileName = Path.Combine("D://PlantarumSamples", fileName);
 8
 9      Stream imgStream = new MemoryStream(GetBytesFromStream(prImage,
10                                         System.Text.Encoding.ASCII));
11
12      SaveTemp(imgStream);
13
14      Image img = Image.FromStream(imgStream);
15      img.Save(fileName);
16      String coordinates = "";
17      try
18      {
19        byte[] coordByes = img.GetPropertyItem(270).Value;
20
21        if (coordByes != null)
22          coordinates = System.Text.Encoding.ASCII
23                                .GetString(coordByes).Substring(0,
24                                System.Text.Encoding.ASCII.GetString(
25                           img.GetPropertyItem(270).Value).Length - 1);
26      }
27      catch (System.ArgumentException)
28      {
29        return null;
30      };
31
32      var specimen = ClassifySample(coordinates, fileName);
33      if (specimen == null)
34        return SaveExif("NotRecognized", img, fileName);
35      else
36        return specimen;
37    }
38    catch (Exception ex)
39    {
40      string erro = "ERR" + ex.Message;
41      if (ex.InnerException != null)
42        erro += "/nUploadFile err inner exception:" +
43                                           ex.InnerException.Message;
44    }
45    return null;
46  }
```

O método exibido no Quadro 13 cria um arquivo em que o nome é a data da chamada, para salvar a imagem localmente antes de enviar para a classificação. Também antes da classificação é necessário ler as propriedades EXIF para obter as informações textuais da imagem. Após isso é feita uma chamada para `ClassifySample` passando o caminho da imagem salva e as informações lidas.

O método `ClassifySample` faz a conexão do WCF com a API Plantarum (CASSANIGA, 2012). Para isso o método recebe o caminho de uma imagem e um texto contendo as coordenadas do pecíolo e ponta da folha. Então é feita a conversão do texto recebido para dois objetos do tipo `Point` e depois é iniciada a comunicação com a API de classificação. O Quadro 14 mostra a implementação deste método.

Quadro 14 – Conectando com a API para a classificação

```
 1  private Stream ClassifySample(string prMetaTags,
 2                                string prImagePath)
 3  {
 4      var dic = GetInformationFromString(prMetaTags);
 5      string pontos = dic["CoordinatesPecioloPonta"];
 6
 7      Point petiole = new Point(int.Parse(pontos.Split(';')[0]
 8                                                  .Split(',')[0]),
 9                                int.Parse(pontos.Split(';')[0]
10                                                  .Split(',')[1]));
11      Point tip = new Point(int.Parse(pontos.Split(';')[1]
12                                              .Split(',')[0]),
13                            int.Parse(pontos.Split(';')[1]
14                                              .Split(',')[1]));
15
16      api = new PlantarumAPI();
17      api.OpenDatabase(DB_PATH);
18      api.LoadClassifier();
19
20      long specimen = 0;
21      specimen = api.Classify(prImagePath, petiole, tip);
22
23      if (specimen == 0)
24          return null;
25
26      var specInfo = api.GetSpecimenInfo(specimen);
27      var imgInfo = api.GetSpecimenImage(specimen);
28
29      if (imgInfo != null)
30      {
31        var memStr = new MemoryStream(imgInfo.Data);
32        var image = Image.FromStream(memStr);
33        return SaveExif(specInfo.Name, image, prImagePath);
34      }else
35        return SaveExif("specInfo:" + specInfo.Name,
36                        Image.FromFile(prImagePath.Substring(0,
37                                       (prImagePath.LastIndexOf(@"\")+1))
38                                                    + "temp.jpg"),
39                         prImagePath);
40  }
```

Como mostra o Quadro 14, os passos necessários para a utilização da API, são a inicialização da API, que se dá pela chamada ao construtor, seguido da abertura da base de dados. Para a classificação da amostra é necessário que seja feita a carga do classificador que foi cadastrado para a base que foi carregada, e só então é possível fazer a classificação. A API retorna o `id` da espécie que foi encontrada, depois com este `id` é solicitado para a API a

imagem e as informações da espécie que foi reconhecida. Se não for possível reconhecer a imagem, o método `Cassify` da API lança uma exceção que é tratada antes de responder ao cliente no método `UploadClassifySample`. Por último, quando a amostra é reconhecida pela API, o nome da espécie é salvo na propriedade EXIF da imagem e a imagem é enviada para o cliente como retorno da classificação.

Para o cadastro de espécie ou amostra, o contrato WCF disponibiliza o método `UploadNewSample`, que recebe uma imagem do tipo `Stream` por parâmetro e executa o método com o mesmo nome na classe que implementa este contrato. Para cadastro de espécies e classificação de espécies tem-se um único método, pois se o usuário desejar cadastrar uma amostra ele deve selecionar uma das já cadastradas na base, e se ele desejar cadastrar uma nova amostra ele deve colocar um nome inexistente na base. O Quadro 15 demonstra a implementação deste método.

Quadro 15 – Recebendo uma imagem cadastro de amostra ou espécie

```
 1  string IPS.UploadNewSample(Stream prImage)
 2  {
 3    try
 4    {
 5      var fileName = string.Format("{0}.jpg", DateTime.Now.ToString(
 6                                                    "yyyyMMddHHmmss"));
 7      fileName = Path.Combine("D://PlantarumSamples", fileName);
 8
 9      Stream imgStream = new MemoryStream(GetBytesFromStream(
10                                prImage, System.Text.Encoding.ASCII));
11
12      Image img = Image.FromStream(imgStream);
13      img.Save(fileName);
14      String dataSample = "";
15      try
16      {
17        byte[] coordByes = img.GetPropertyItem(270).Value;
18
19        if (coordByes != null)
20          dataSample = Encoding.ASCII.GetString(coordByes).Substring(0,
21  System.Text.Encoding.ASCII.GetString(img.GetPropertyItem(270)
22                                                    .Value).Length - 1);
23      }
24      catch (System.ArgumentException)
25      {
26        return null;
27      };
28
29      RegisteringSpecimen(dataSample, fileName, prImage);
30
31      return "OK";
32    }
33    catch (Exception ex)
34    {
35      string erro = "UploadFile err:" + ex.Message;
36      if (ex.InnerException != null)
37        erro += "/nUploadFile err inner exception:"
38                                          + ex.InnerException.Message;
39    }
40    return null;
41  }
```

O método `UploadNewSample`, da mesma forma que o `UploadClassifySample`, recebe uma imagem no formato `Stream`, salva-a e extrai as características que estão na propriedade EXIF. Após isso efetua uma chamada para `RegisteringSpecimen` onde é cadastrada uma nova amostra ou uma nova espécie. O Quadro 16 mostra a implementação deste método.

Quadro 16 – Cadastrando uma nova amostra ou nova espécie

```
 1  private void RegisteringSpecimen(string prData, string prImagePath,
 2                                                          Stream prImage)
 3  {
 4      var dic = GetInformationFromString(prData);
 5      string specimenName = dic["SpecimenName"];
 6      string specimenComment = dic["SpecimenComment"];
 7      string useImageToSpecimen = dic["UseImageToSpecimen"];
 8      string pontos = dic["CoordinatesPecioloPonta"];
 9
10      if (specimenName == "" || pontos == "")
11        throw new Exception(@"CoordinatesPecioloPonta ou
12                        SpecimenName estão vazios e são obrigatórios");
13
14      api = new PlantarumAPI();
15      api.OpenDatabase(DB_PATH);
16
17      var specimens = api.GetSpecimens();
18      bool contains = false;
19      foreach (var spec in specimens)
20      {
21        if(spec.Name == specimenName)
22          contains = true;
23      }
24
25      if (!contains){
26        api.AddSpecimen(new PlantarumAPI.SpecimenInfo(){
27                                                  Name = specimenName});
28      }
29
30      specimens = api.GetSpecimens();
31      Dictionary<string, long> specDic = new Dictionary<string,long>();
32      foreach (var spec in specimens)
33      {
34        specDic.Add(spec.Name, spec.ID);
35      }
36
37      Point petiole = new Point(int.Parse(pontos.Split(';')[0]
38                                                  .Split(',')[0]),
39                                int.Parse(pontos.Split(';')[0]
40                                                  .Split(',')[1]));
41      Point tip = new Point(int.Parse(pontos.Split(';')[1]
42                                              .Split(',')[0]),
43                            int.Parse(pontos.Split(';')[1]
44                                              .Split(',')[1]));
45
46      api.AddSample(specDic[specimenName], prImagePath, petiole,
47                                                  tip, specimenComment);
48      if (useImageToSpecimen == "true"){
49        PlantarumAPI.SpecimenImageInfo info = new
50                                      PlantarumAPI.SpecimenImageInfo();
51        var img = Image.FromFile(prImagePath);
52        ImageConverter converter = new ImageConverter();
53        info.Data = (byte[])converter.ConvertTo(img, typeof(byte[]));
54        info.MimeType = null;
55        api.SetSpecimenImage(specDic[specimenName], info);
56      }
57  }
```

No método `RegisteringSpecimen`, a aplicação recebe a imagem e os dados textuais informados na aplicação cliente e então separa as informações que foram digitadas para verificar se o nome da espécie já existe na base de dados. Quando o nome já existir na base de dados, somente é cadastrada uma amostra para a espécie encontrada. Quando não existir, é cadastrada primeiramente uma nova espécie, e então é adicionada uma amostra para a espécie.

3.3.1.3 Implementação de novas características na API

A API Plantarum de Cassaniga (2012) foi construída utilizando como plataforma as linguagens C e C++, oferecidas pela plataforma Microsoft visual C++ no ambiente de desenvolvimento Microsoft Visual Studio 2010. Para a segmentação de imagens Cassaniga (2012) utilizou a bilioteca de código aberto OpenCV.

Para a persistência de dados Cassaniga (2012) fez uso da biblioteca SQLite em conjunto com a biblioteca CppSQLite. Estas bibliotecas foram escolhidas por possibilitarem acesso aos dados com uma abordagem orientada à objetos e por não precisarem de instalação de nenhum serviço para sua utilização.

Foram implementadas três novas características para classificação de espécies a serem informadas pelo usuário, as quais foram retiradas do estudo de Saueressig (2012 p. 14-16) onde ele as expõe como características a fim de cobrir variações morfológicas entre os indivíduos. Foram escolhidas para a implementação deste trabalho as características de tipo da folha, pilosidade e discoloridade, possibilitando ao usuário informar se a amostra a ser classificada é composta, possui ou não pelos, e se o verso da amostra possui cor diferente da parte fotografada.

Para isto, foi necessária a inclusão de três novas classes na API e também fez-se necessária a inclusão de novos parâmetros para os métodos `PLNT_DB_add_sample` e `PLT_classify` que faziam adição de espécies e classificação respectinamente. As classes incluídas extendem da interface IFeature criada por Cassaniga (2012), portanto os métodos para a persistência inclusão na base de dados e utilização destas características pela rede neural não precisaram ser alterados.

3.3.2 Operacionalidade da implementação

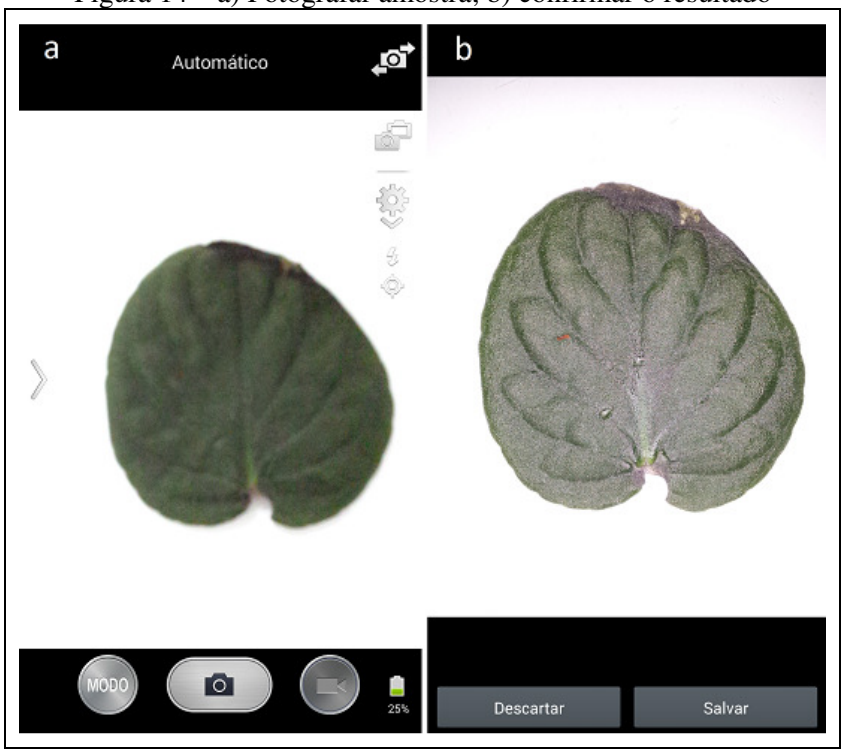
Esta seção é destinada a mostrar o funcionamento da aplicação. A Figura 13 mostra a tela inicial do aplicativo, que possibilita escolher entre três ações, fotografar para classificar, cadastrar uma nova amostra ou enviar uma imagem salva para a classificação.

Figura 13 – Tela inicial da aplicação

Nesta tela são apresentados três botões e uma imagem de abertura. O botão à esquerda inicia o processo para fotografar uma imagem e enviar ao servidor para classificação. O botão do meio chama as funções para o cadastro de uma nova espécie na base de dados do servidor. O botão à direita tem como função permitir que seja selecionada uma imagem para ser enviada para a classificação.

Ao tocar no botão à esquerda o usuário é direcionado para a câmera para fotografar uma amostra e enviá-la ao servidor. A Figura 14 mostra o estado da aplicação após tocar no botão à esquerda.

Figura 14 – a) Fotografar amostra; b) confirmar o resultado

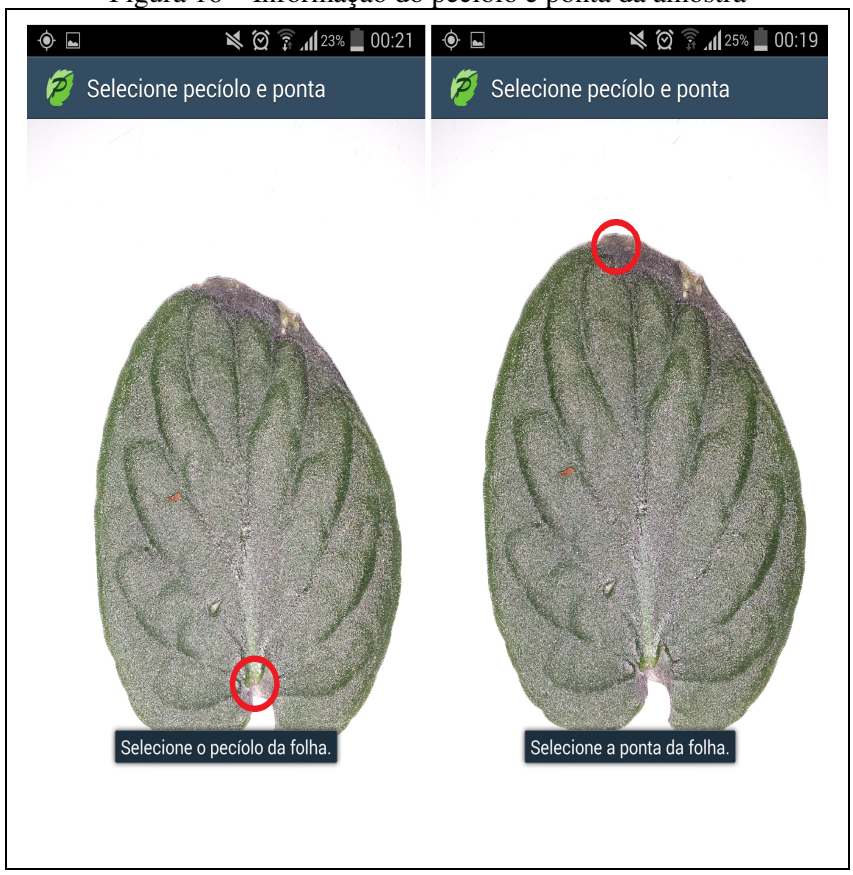


Após a imagem adquirida é solicitado que o usuário insira as informações de tipo da folha, pilosidade, discoloridade e possui uma opção para caso estas informações devam ou não ser desconsideradas. A Figura 15 mostra a tela responsável por esta parte do processo.

Figura 15 – Informações da amostra

Nesta tela o usuário seleciona dentre as opções disponíveis as que correspondem com a amostra que ele está submetendo para análise. Após tocar no botão OK ele é direcionado para uma tela onde deverão ser informados o pecíolo e a ponta da folha. A Figura 16 retrata como é feito este processo.

Figura 16 – Informação do pecíolo e ponta da amostra



Neste processo o usuário informa as coordenadas tocando nas partes correspondentes da folha. Ao tocar a primeira vez na parte correspondente ao pecíolo a aplicação guarda as coordenadas e solicita que o usuário toque na parte correspondente a ponta da folha. As circunferências representam os toques utilizados para passar por esta parte do processo. Após tocar a ponta da folha, uma mensagem é exibida para saber se o usuário deseja manter os pontos, inverter pecíolo e ponta, ou descartar.

Ao optar por manter os pontos a aplicação retorna para a tela inicial e solicita uma nova interação para decidir se deve salvar a imagem, enviar ou descartar. Caso o usuário opte por salvar ou por descartar a imagem a aplicação salvará e retornará a tela principal, caso contrário, optando por enviar, a aplicação envia a imagem e espera o retorno do servidor.

Quando o servidor responde se a imagem foi classificada, o resultado é mostrado na tela principal permitindo que o usuário possa repetir o processo sem precisar executar qualquer outra ação. A Figura 17 mostra o resultado da classificação quando positivo.

Figura 17 – Resultado com sucesso da classificação

O processo de fotografar uma imagem e enviá-la para o servidor compreende o processo mais longo e no caso de teste onde a amostra não foi reconhecida, compreende também a funcionalidade para adicionar amostra. Quando uma amostra não é reconhecida, caso o usuário deseje cadastrar uma nova espécie, a aplicação utiliza a funcionalidade acionada pelo botão adicionar espécie, alterando o fato de que não será necessária a seleção de uma imagem, pois o cadastro da espécie será feito com a própria imagem enviada ao servidor.

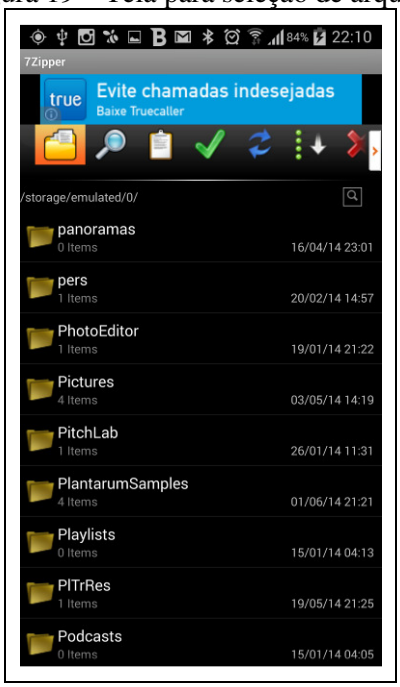
Ao obter um resultado negativo da classificação, é exibida uma mensagem para informar que a imagem não foi reconhecida e verificar se o usuário deseja ou não cadastrar uma nova espécie. Caso seja escolhido cadastrar a nova espécie o usuário é direcionado a uma tela que contém campos para o preenchimento das informações da espécie e mostra qual será a imagem da espécie a ser cadastrada. A Figura 18 mostra a tela de cadastro de espécies.

Figura 18 – Tela de cadastro de espécies

A tela representada pela Figura 18 mostra os campos que o usuário deve informar para o cadastro da amostra. O campo `Espécie da amostra` é do tipo auto completar, na medida em que o usuário digita, a aplicação sugere as espécies já cadastradas na base de dados. Isto possibilita que se a amostra for de uma nova espécie esta seja cadastrada e se for apenas mais uma amostra de uma espécie já existente seja cadastrada apenas a amostra.

Outro caminho para chegar nesta tela é o botão do meio da tela inicial, porém, ao utilizar este caminho é solicitado que o usuário selecione a imagem previamente salva que deseja cadastrar. A tela apresentada na Figura 19 pode variar, pois a tela para seleção de imagem varia conforme o gerenciador de arquivos que estiver instalado e configurado como padrão no dispositivo. No dispositivo utilizado para teste o gerenciador era o 7Zipper, portanto a Figura 19 mostra a tela de seleção de arquivos pelo 7Zipper.

Figura 19 – Tela para seleção de arquivo



As imagens salvas pelo aplicativo ficam dentro da pasta `PlantaumSamples` que será criada assim que o aplicativo for instalado. Após selecionar uma imagem, antes de chegar à tela de cadastro de amostra, é verificado se a imagem já possui as características textuais informadas pelo usuário, e se já possui as coordenadas de pecíolo e ponta. Ao constatar que a imagem não possui características textuais a aplicação abre a tela exibida na Figura 15 para que o usuário as informe. Se a imagem não possuir as coordenadas de pecíolo e ponta, a aplicação solicita que o usuário as informe pela tela exibida na Figura 14.

Após assegurar que estas informações estão preenchidas, a aplicação verifica então se a imagem possui coordenadas de GPS e, se não possuir, a própria aplicação as preenche. Só então se dá continuidade no fluxo e vai para a tela de cadastro de espécies. A partir do cadastro de espécie preenchido e confirmado, a aplicação envia esta imagem para o servidor cadastrá-la na base como espécie ou amostra e depois retorna para a tela inicial informando se a imagem foi cadastrada com sucesso ou se houve algum erro.

A última funcionalidade do sistema a ser explicada é disparada pelo botão da direita na tela inicial. Este botão tem a função de enviar uma imagem salva no dispositivo para a classificação. Ao tocar neste botão é aberta a tela retratada na Figura 19 e após a seleção do arquivo, da mesma forma que acontece no cadastro de amostra, o sistema verifica as informações textuais, coordenadas de pecíolo e ponta. Na falta de alguma característica, a aplicação abre a tela correspondente para o preenchimento desta. Após isto é feita a verificação de coordenadas GPS, se estiver inexistente a própria aplicação preenche. Ao final deste processo a imagem é enviada ao servidor para classificação, e se o resultado for positivo

exibe a tela representada pela Figura 13. Se o resultado for negativo, questiona se o usuário deseja cadastrar uma nova amostra ou não da mesma forma que acontece no final do processo de fotografar imagem e enviar para classificação.

3.4 RESULTADOS E DISCUSSÃO

Esta seção é dedicada à comprovação da funcionalidade da aplicação ao fotografar e reconhecer as espécies de plantas e para isto foram efetuados dois cenários distintos de teste. O primeiro cenário foi feito utilizando imagens impressas de um banco de imagens, enquanto o segundo cenário foi construído com espécies locais.

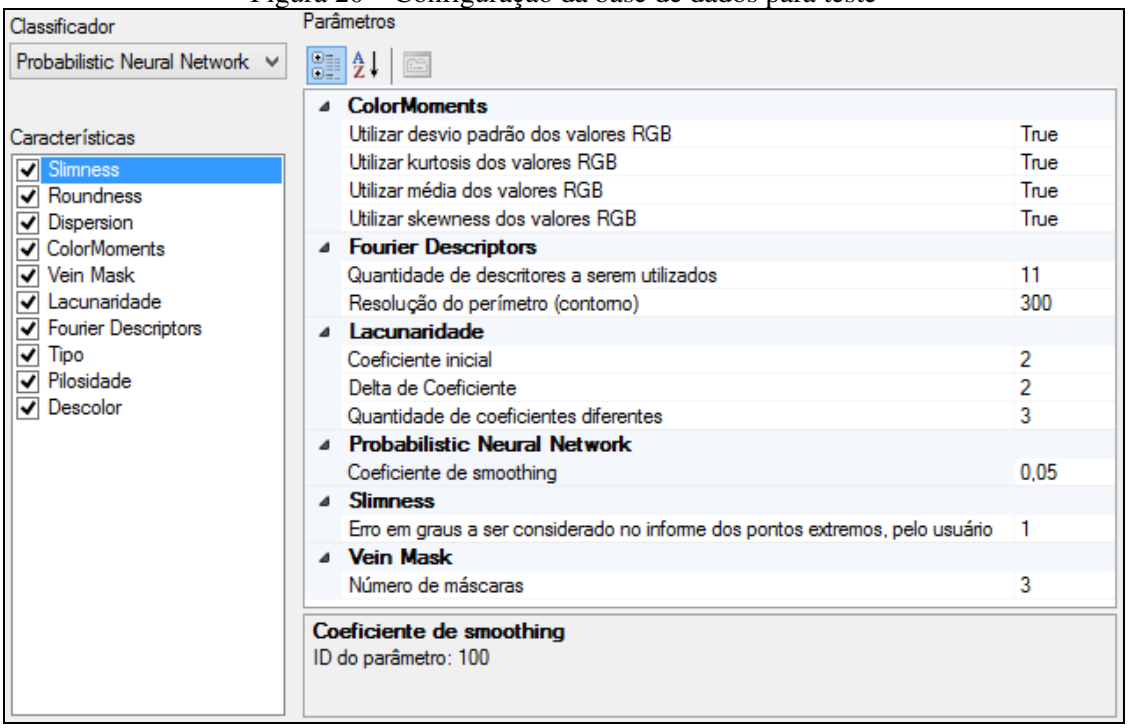
Esta seção foi dividida para explicar cada um dos contextos, onde a subseção 3.4.1 trata os testes com o banco de imagens e a subseção 3.4.2 exibe os testes com amostras locais. No final desta seção há uma área que compara os resultados deste trabalho com os trabalhos correlatos, descrita pela subseção 3.4.3.

3.4.1 Testes com banco de imagens

No cenário fictício foi utilizado o banco de imagens Flavia *dataset*, disponibilizado gratuitamente por Wu et al. (2007). Neste banco de imagens, as espécies não possuíam as características que foram incluídas na API, portanto para os testes estes aspectos foram simulados marcando as espécies com atributos que não necessariamente possuíam.

Como foram incluídos três novos atributos e cada um destes atributos possui um dos valores, a maior combinação distinta destes atributos é de oito diferentes entre si. Por isso para os testes foram selecionadas oito espécies, atribuindo para elas uma destas oito combinações. Utilizando a ferramenta disponibilizada por Cassaniga (2012), foi criada uma base de dados com a parametrização recomendada por ele na página 89 de seu trabalho, e apta a salvar as novas características. A Figura 20 mostra as configurações utilizadas para a criação da base de dados na aplicação disponibilizada por Cassaniga (2012).

Figura 20 – Configuração da base de dados para teste



Para a população deste banco de dados foi utilizada uma ferramenta console disponibilizada por Cassaniga (2012) para o treino da rede neural. Cada espécie teve um treino separado com todas as amostras presentes no banco de imagens Flávia *dataset*. Neste treino foi necessário alterar o código fonte da ferramenta para que cada uma das espécies fosse incluída com uma combinação diferente das novas características implementadas.

Ao fim da construção da base de dados foram impressas quatro amostras de cada espécie para testar a aplicação, as amostras impressas podem ser consultadas no Anexo A. Ao iniciar os testes foram encontrados problemas no reconhecimento por fatores externos como, por exemplo, iluminação, configuração da câmera do dispositivo e foco da imagem a ser processada.

A partir destes problemas identificados foi iniciada a primeira bateria de testes com as amostras impressas, utilizando condições adequadas de iluminação, atentando-se para as configurações da câmera e foco da imagem. Os resultados obtidos foram positivos, sendo que de trinta e duas imagens submetidas para classificação, apenas duas não foram reconhecidas e quatro tiveram falso positivo, indicando uma espécie incorreta. Para a segunda bateria de testes as imagens que não haviam sido reconhecidas foram adicionadas à base de dados e as que possuíam falso negativo também, e então ao refazer o teste apenas uma imagem acusou falso positivo e nenhuma ficou sem ser reconhecida.

Uma terceira bateria de testes foi feita com a base de dados utilizando uma ferramenta disponibilizada por Cassaniga (2012). Nesta bateria foi construída uma nova base de testes

onde foram adicionadas as mesmas espécies, porém apenas setenta por cento das amostras para cada espécie foi inserida. As espécies que não foram inseridas foram submetidas à classificação. Os resultados podem ser vistos na Tabela 1, que lista as espécies utilizadas para a validação da aplicação.

Tabela 1 – Espécies de plantas utilizadas na validação da aplicação

Espécie	Número de amostras de teste	Número de amostras reconhecidas	Percentual de acerto
Acer Palmatum	16	16	100%
Aesculus chinensis	17	16	94.12%
Berberis anhweiensis Ahrendt	18	18	100%
Cercis chinensis	22	22	100%
Indigofera tinctoria L.	19	19	100%
Kalopanax septemlobus (Thunb. ex A.Murr.) Koidz.	17	17	100%
Phoebe nanmu (Oliv.) Gamble	20	19	95%
Phyllostachys edulis (Carr.) Houz.	16	16	100%
Total	146		98,64%

O resultado mostra que as características contribuíram no reconhecimento das amostras. Em comparação com os resultados da versão anterior da API a nova versão mostra um percentual maior de acerto, como apresentado na Tabela 2.

Tabela 2 – Comparação dos resultados de classificação entre as versões da API

Espécie	Número de amostras de teste API 1.0.0	Número de amostras reconhecidas API 1.0.0	Percentual de acerto API 1.0.0	Numero de amostras de teste API 1.1.0	Número de amostras reconhecidas API 1.1.0	Percentual de acerto API 1.1.0
Acer Palmatum	16	16	100%	16	16	100%
Aesculus chinensis	22	22	100%	17	16	94.12%
Berberis anhweiensis Ahrendt	17	17	100%	18	18	100%
Cercis chinensis	18	18	100%	22	22	100%
Indigofera tinctoria L.	19	18	94.75%	19	19	100%
Kalopanax septemlobus (Thunb. ex A.Murr.) Koidz.	20	20	100%	17	17	100%
Phoebe nanmu (Oliv.) Gamble	17	17	100%	20	19	95%
Phyllostachys edulis (Carr.) Houz.	16	13	81.25%	16	16	100%
Total	145		97.24%	146		97.94%

O cenário para os testes foi construído da mesma forma para as duas versões e as amostras para cadastro e treino são geradas aleatoriamente pela ferramenta disponibilizada por Cassaniga (2012).

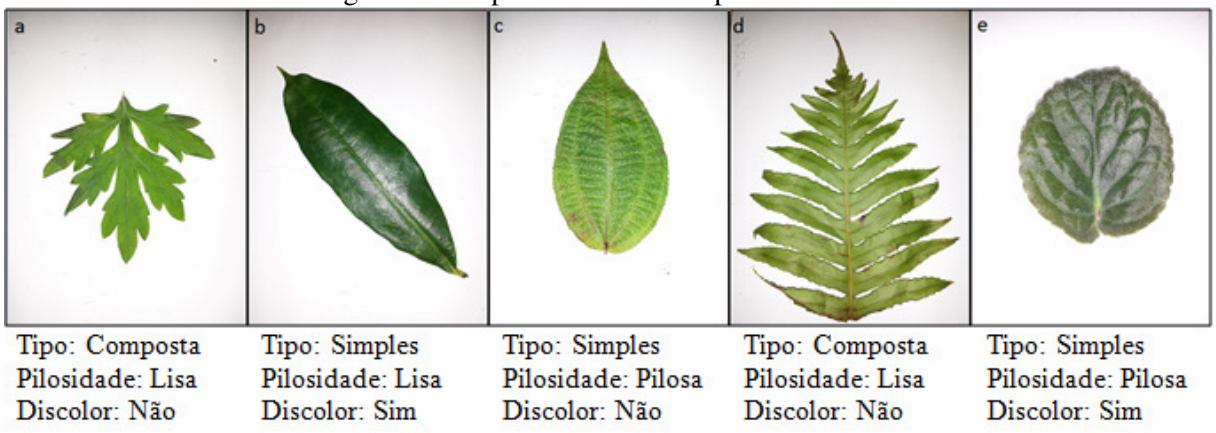
3.4.2 Testes com espécies locais

Para o segundo cenário foram utilizadas espécies reais de plantas, selecionando algumas espécies que possuíam os atributos que foram incluídos na API. O processo de população da base de dados e o treinamento da rede neural aconteceram unicamente por intermédio da aplicação desenvolvida, simulando com maior veracidade o cenário de utilização da aplicação.

Para os testes em ambiente real foram encontrados problemas de iluminação e contraste na imagem, causados pela sombra gerada pela própria amostra e pela falta de iluminação de exposição da imagem. Para contornar o problema de contraste foi necessário que as amostras fossem fotografadas sobre um vidro sobre uma superfície branca. Para contornar o problema da sombra da amostra optou-se por utilizar o flash do dispositivo, obtendo resultados positivos.

Foram utilizadas cinco diferentes espécies para esta seção de testes, utilizando diferentes combinações das novas características incluídas na API. A Figura 21 mostra as espécies cadastradas.

Figura 21 – Espécies cadastradas para testes reais



A Figura 21 mostra uma amostra de cada espécie cadastrada, em ordem as características utilizadas foram: a) composta, b) discolor, c) pilosa, d) composta e discolor, e) pilosa e discolor. As espécies de fato possuíam as características indicadas no momento do cadastro.

Para iniciar os testes de classificação foram incluídas nove amostras de cada uma das espécies, sempre alterando a posição da ponta da folha. Após o cadastro foram fotografadas sete amostras de cada espécie e submetidas à classificação.

A maioria das amostras foi bem sucedida na classificação. A `Espécie b` apresentou falso positivo com a espécie como `Espécie c`, mais de uma vez ao submeter uma amostra

com coloração mais clara. As demais espécies tiveram resultados positivos ao serem submetidas com as características informadas corretamente.

Foi realizada uma segunda seção de testes com esta mesma base de dados, no qual as características de textura, cor e formato não foram informadas ao submeter às imagens para classificação. Em nenhum dos casos a espécie foi reconhecida com sucesso, e ao submeter a `Espécie e` sem as características aconteceu falso positivo com a `Espécie c`.

Com estes testes é possível perceber que as novas características implementadas na API interferem no resultado do reconhecimento de forma significativa. Ao omitir os detalhes que foram incluídos na classificação o reconhecimento falhou em todas as tentativas, porém ao se informar as características devidamente os resultados foram positivos na sua maioria identificando as espécies com sucesso.

3.4.3 Comparação com os trabalhos correlatos

O Quadro 17 apresenta a comparação entre algumas características dos trabalhos relacionados com o presente trabalho.

Quadro 17 – Comparativo dos trabalhos correlatos com este projeto

Características /Trabalhos relacionados	Knight, Painter e Potter (2010)	Kumar et al. (2012)	Plantarum (2014)
Coordenadas de GPS	Não	Sim	Sim
Extração das características	Dispositivo	Servidor	Servidor
Características utilizadas	Geométricas	Geométricas	Geométricas, nervuras, descritores de Fourier, cor, textura, e descrição de aspecto informado pelo usuário
Método de reconhecimento	Automático	Com auxilio do usuário	Automático
Plataforma	Android	iOS	Android
Precisão média	80	96,8	97.94

A proposta apresentada por Sangle, Shirsat e Bhosle (2013) não é apresentada no Quadro 17, pois não consta no trabalho a descrição da precisão média, impossibilitando a comparação com os demais.

Entre os trabalhos de Kinght Painter e Potter (2010) e Kumar et al. (2012) a diferença existe porque a cada classificação o trabalho de Kumar et al. (2012) retorna uma lista com os vinte possíveis resultados, enquanto Kinght Painter e Potter (2010) retornam apenas a espécie com o acerto.

É possivel notar um aumento na precisão ao comparar os trabalhos que utilizam apenas características geométricas para a classificação com o presente trabalho, que utiliza

caracteristicas geométricas, descritores de Fourrier, cor, textura, e ainda a descrição de três características informadas pelo usuário para determinar o tipo, formato e a textura da amostra.

4 CONCLUSÕES

Este trabalho apresentou o desenvolvimento de uma aplicação móvel para captura de imagens de folhas de plantas, informações de localização e de características morfológicas através de um *smartphone* com sistema operacional Android. Também propôs o desenvolvimento de uma aplicação servidora capaz de processar a informação e fazer a classificação de espécies de plantas por meio de uma API. Uma espécie pode ser reconhecida ou adicionada passando para o servidor a imagem da folha da planta com coordenadas de pecíolo e ponta da folha e também descrevendo algumas características da folha em questão.

A adição de três novas características foi importante, pois além de aumentar o índice de acerto nos testes da API, possibilitou que fossem obtidos bons resultados nos testes com espécies reais.

Para os testes do aplicativo cliente foi utilizado um Samsung Galaxy S IV com sistema Android versão 4.4.2, que possibilitou atingir o objetivo com sucesso devido à qualidade da câmera do dispositivo.

A implementação do aplicativo para servidor teve um bom resultado ao ser utilizado a linguagem C#, porém teria sido mais proveitoso e facilitaria a integração com a API se tivesse sido desenvolvido com o Microsoft Visual Studio 2010, pois o compilador de C++ do Microsoft Visual Studio 2010 é diferente da versão 2012 do Microsoft Visual Studio. Isto gerou uma série de problemas que poderiam ter sido evitados ao utilizar a mesma versão em que foi contruida a API.

O maior problema ao incluir as características na API foi utilizar o Microsoft Visual Studio 2012 para isto. Como a API foi construída no Microsoft Visual Studio 2010, suas dependências OpenCV e QT necessitavam do compilador desta versão. Foi instalada a versão 2010 do Microsoft Visual Studio e mesmo selecionando a versão correta do compilador para a API, ao compilar acontecia um erro que impossibilitava a criação dos binários. Foi necessária a criação de uma máquina virtual para a compilação da API onde só foi instalada a versão 2010 do Microsoft Visual Studio, e assim foi possível a criação dos binários da API.

O desenvolvimento deste trabalho possibilita o reconhecimento e a classificação de espécies baseado no registro de imagens de suas folhas, facilitando o trabalho por diminuir a dependência de catálogos impressos ou conhecimento das caracteristiccas determinantes de determinada espécie.

4.1 LIMITAÇÕES

A aplicação desenvolvida possui as seguintes limitações:

a) o processo depende um uma imagem de entrada com alto contraste entre a folha e o fundo sem ruídos;
b) a aplicação cliente não define automaticamente as configurações da câmera;
c) a aplicação cliente somente executa em telas com resolução de 1920x1080 pixels;
d) a imagem de entrada deve ser fotografada com o dispositivo em orientação retrato;
e) a aplicação depende de uma base de dados previamente criada para a API Plantarum;
f) para o cadastro e identificação de espécies é necessário que a amostra tenha sido fotografada e salva pela própria aplicação.

4.2 EXTENSÕES

Algumas sugestões deste trabalho são listadas a seguir:

a) implementação de novas características a serem informadas pelo usuário;
b) permitir a utilização da câmera em qualquer orientação;
c) portar a aplicação à dispositivos com diferentes tamanhos de tela;
d) fazer um pré-tratamento da imagem de entrada no dispositivo a fim de eliminar a necessidade do alto contraste;
e) permitir enviar para a classificação e cadastro qualquer imagem do dispositivo;
f) criar mecanismo na aplicação servidor que ao notar a não existência de uma base de dados configurada, faça a criação da mesma com as configurações recomendadas;
g) implementar interface para o serviço criado para outras plataformas de dispositivos móveis;
h) implementar a na aplicação cliente as configurações recomendadas para a câmera.

REFERÊNCIAS

BORGES Jr, M. P. **Desenvolvendo webservices**. Rio de Janeiro: Ciência Moderna, 2005.

CASSANIGA, A. W. **PLANTARUM:** API para reconhecimento de plantas. 2012. 113 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau. Disponível em: <http://campeche.inf.furb.br/tccs/2012-II/TCC2012-2-04-VF-ArnoWCassaniga.pdf>. Acesso em: 07 set. 2013.

FERREIRA, Flávio M. S. **Home gateway do living usability lab**. 2012. 118 f. Dissertação – Departamento de Eletrônica, Telecomunicações e Informática, Universidade de Aveiro, Aveiro. Disponível em: <http://ria.ua.pt/bitstream/10773/10264/1/Disserta%C3%A7%C3%A3o.pdf>. Acesso em: 07 set. 2013.

FERRI, Mário G. **Botânica**: Morfologia externa das plantas. São Paulo: Nobel, 1981.

GOMES, Evandro W. C. **Global interface process usando android através de webservices**. 2011. 47 f. Monografia de Especialização (Especialista em Tecnologia Java) – Departamento Acadêmico de Informática, Universidade Tecnológica Federal do Paraná, Curitiba. Disponível em: <http://repositorio.roca.utfpr.edu.br/jspui/bitstream/1/619/1/CT_JAVA_VI_2010_05.PDF>. Acesso em: 07 set. 2013.

KNIGHT, David; PAINTER, James; POTTER, Matthew **Automatic plant leaf classification for a mobile field guide:** an android application. 2010. 5 f. Departamento de Engenharia Elétrica - Universidade de Stanford, Stanford, Califórnia. Disponível em: <http://www.stanford.edu/class/ee368/Project_10/Reports/Knight_Painter_Potter_PlantLeafClassification.pdf>. Acesso em: 07 set. 2013.

KUMAR, Neeraj; et al. **Leafsnap:** a computer vision system for automatic plant species identification. 2012. 14 f. Universidade de Washington, Seattle, Washington. Disponível em: <http://homes.cs.washington.edu/~neeraj/papers/nk_eccv2012_leafsnap.pdf>. Acesso em: 07 set. 2013.

PLOTZE, Rodrigo O. **Identificação de espécies vegetais através da análise da forma interna de órgãos foliares**. 2004. 152 f. Dissertação (Mestrado em Ciências de Computação e Matemática Computacional) - Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Paulo. Disponível em: <http://www.teses.usp.br/teses/disponiveis/55/55134/tde-03052006-034021/publico/ME_RodrigoPlotze_ICMC_2004.pdf>. Acesso em: 07 set. 2013.

SANGLE, Shailesh; SHIRSAT, Kavita; BHOSLE, Varsha. **Shape based plant leaf classification system using Android**. 2013. 8 f. Disponível em: <http://www.ijert.org/browse/volume-2-2013/august-2013-edition?download=4872%3Ashape-based-plant-leaf-classification-system-using-android&start=250 >. Acesso em: 07 set. 2013.

SAUERESSIG, Daniel. **Levantamento dendrológico na floresta ombrófila**. 2012. 131 f. Dissertação (Mestrado em Ciências Florestais) - Setor de Ciências Agrárias, Universidade Federal do Paraná, Curitiba. Disponível em: <http://www.floresta.ufpr.br/pos-graduacao/defesas/pdf_ms/2012/d618_0783-M.pdf>. Acesso em: 20 maio 2013.

WIGGERS, Ivonei; STANGE, Carlos E. B. **Manual de instruções para coleta, identificação e herborização de material botânico**. 2008. Disponível em <http://www.diaadiaeducacao.pr.gov.br/portals/pde/arquivos/733-2.pdf>. Acesso em: 29 jun 2014.

WU, Stephen G. et al. A leaf recognition algorithm for plant classification using probabilistic neural network. In: IEEE INTERNATIONAL SYMPOSIUM ON SIGNAL PROCESSING AND INFORMATION TECHNOLOGY, 7th, 2007, Giza. **Proceedings**… [S.l]: IEEE, 2008. p. 11-16.

APÊNDICE A – Imagens utilizadas para a validação da aplicação.

Espécies escolhidas do banco de dados Flávia *dataset* para a validação manual da aplicação. Antes de cada espécie a legenda detalha os valores separados por ponto e vírgula que representam consecutivamente o intervalo de amostras da espécie, o rótulo da espécie, o nome científico, o nome popular, e as novas características incluídas. Cada amostra de tem a sua identificação à esquerda.

Intervalo	Rótulo	Nome Científico	Nome Popular	Novas carácteristicas
3282-3334	28	Acer buergerianum Miq	Tridente maple	lisa, simples, não discolor
3282				
3296				

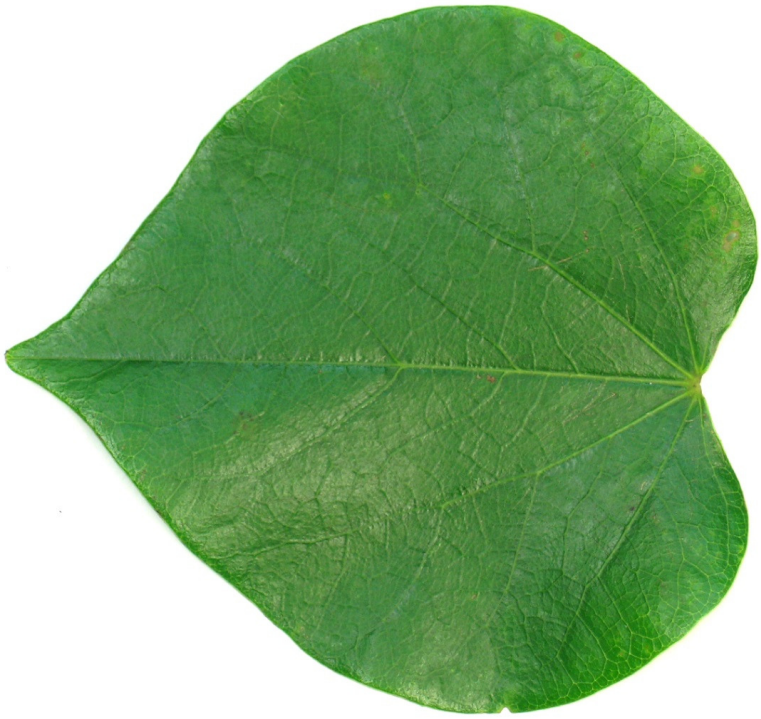
3306	
3301	

Intervalo	Rótulo	Nome Científico	Nome Popular	Novas carácteristicas
1268-1323	6	Acer Palmatum	Japanese maple	lisa, simples, discolor
1273				
1288				

1304	
1319	

Intervalo	Rótulo	Nome Científico	Nome Popular	Novas características
2231-2290	16	Viburnum awabuki K.Koch	Japan Arrowwood	lisa, composta, não discolor
2231				
2244				

2253	
2264	

Intervalo	Rótulo	Nome Científico	Nome Popular	Novas características
1123-1194	4	Cercis chinensis	Chinese redbud	lisa, composta, discolor
1123				
1130				

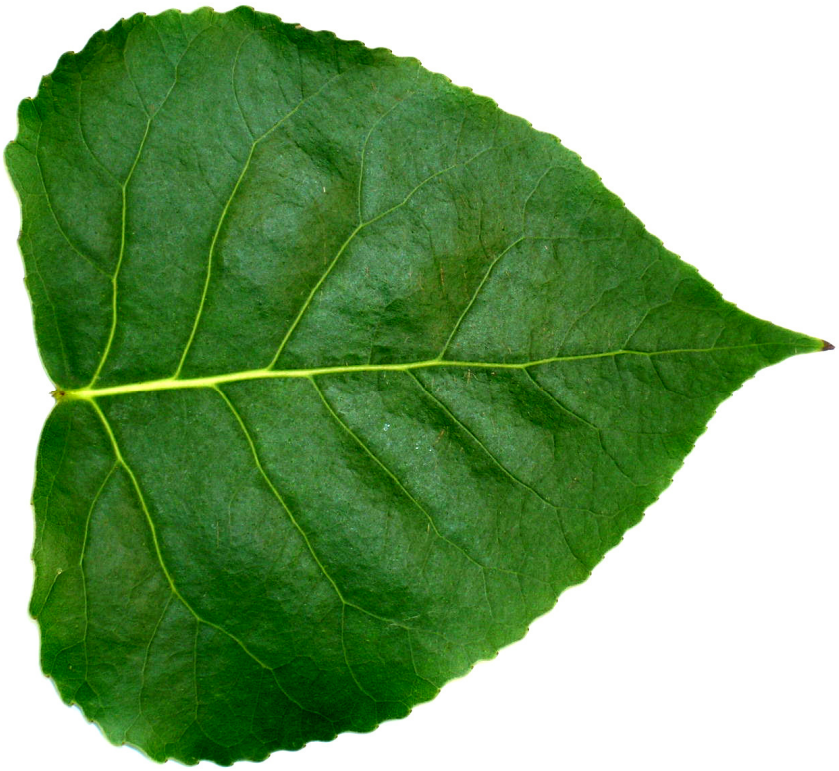
1140	
1185	

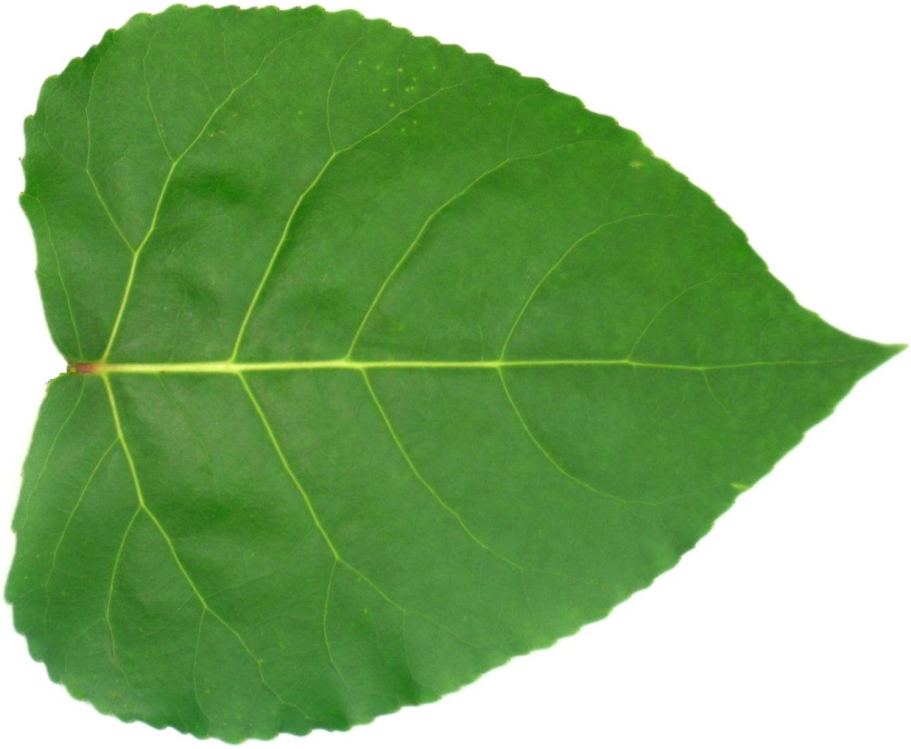
Intervalo	Rótulo	Nome Científico	Nome Popular	Novas características
2424-2485	19	Ginkobiloba L.	ginkgo, maidenhair tree	pilosa, simples, não discolor
2425				
2430				

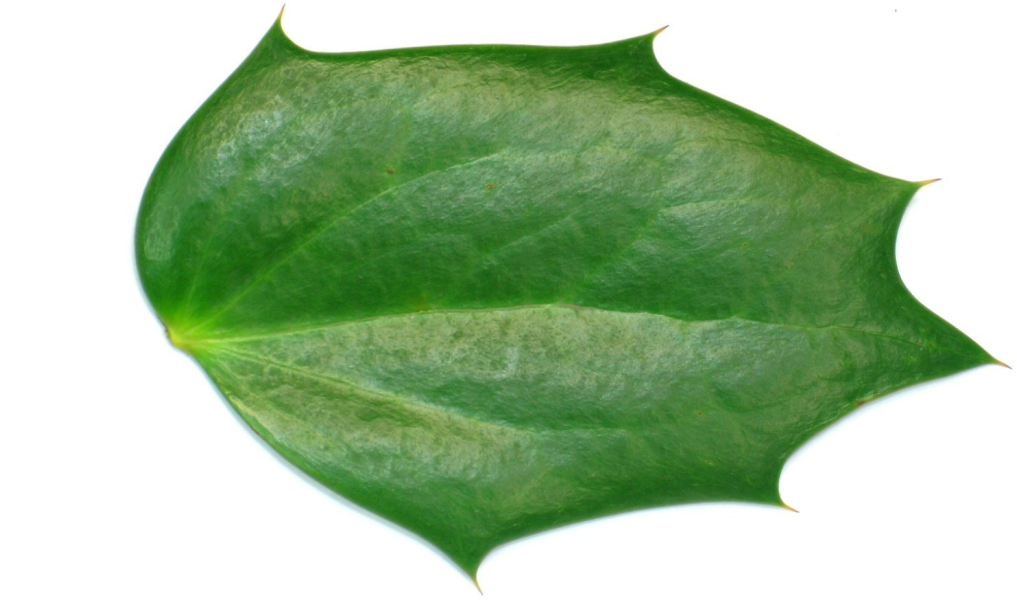
2442	
2469	

Intervalo	Rótulo	Nome Científico	Nome Popular	Novas características
3511-3563	32	Liriodendron chinense (Hemsl.) Sarg	Chinese tulip tree	pilosa, simples, discolor
3511				
3518				

3529	
3550	

Intervalo	Rótulo	Nome Científico	Nome Popular	Novas características
3447-3510	31	Populus× canadensis Moench	Canadian poplar	pilosa, composta, não discolor
3447				
3461				

3471	
3492	

Intervalo	Rótulo	Nome Científico	Nome Popular	Novas características
3335-3389	29	Mahonia bealei (Fortune) Carr.	Beale's barberry	pilosa, composta, discolor
3335				
3354				

3365	
3383	